Inverse Optimization Theory and Applications to Routing Problems

Zattoni Scroccaro, P.

DOI
10.4233/uuid:271a3cb4-8c0c-4294-8834-0313ee4969d9

Publication date
2024

Document Version
Final published version

Citation (APA)
Zattoni Scroccaro, P. (2024). *Inverse Optimization Theory and Applications to Routing Problems*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:271a3cb4-8c0c-4294-8834-0313ee4969d9

Inverse Optimization Theory and Applications to Routing Problems

INVERSE OPTIMIZATION THEORY AND APPLICATIONS TO ROUTING PROBLEMS

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus Prof. dr. ir. T.H.J.J. van der Hagen
chair of the Board of Doctorates
to be defended publicly on
Wednesday 16 October 2024 at 15:00 o'clock

by

Pedro ZATTONI SCROCCARO

Master of Science in Systems and Control,
Delft University of Technology, The Netherlands
born in Curitiba, Brazil.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Dr. P. Mohajerin Esfahani,	Delft University of Technology, promotor
Dr. B. Atasoy,	Delft University of Technology, promotor
Independent members:	
Prof. dr. M. Bierlaire,	Ecole Polytechnique Fédérale de Lausanne, Switzerland
Prof. dr. B. De Schutter,	Delft University of Technology
Prof. dr. D. den Hertog,	University of Amsterdam
Prof. dr. F. Kılınç-Karzan,	Carnegie Mellon University, USA
Prof. dr. T. Keviczky,	Delft University of Technology, reserve member

The work in this thesis has been supported by the TU Delft cohesion grant "Artificial Intelligence for Sustainable Real-time Transportation Systems" and has been partially supported by the ERC grant TRUST-949796.

Keywords: Inverse optimization, convex optimization, first-order algorithms, routing problems, decision-making, behavior modeling

If anyone can refute me—show me I'm making a mistake or looking at things from the wrong perspective—I'll gladly change. It's the truth I'm after, and the truth never harmed anyone. What harms us is to persist in self-deceit and ignorance.

Marcus Aurelius

CONTENTS

SUMMARY

This thesis concerns the fundamental problem of learning the behavior of decision-making agents using only observations of how they act in different situations. As humans, we do it all the time, and have been doing it since birth: think about how a child learns to speak and walk. However, this thesis does not only focus on imitating. We go a step further and try to learn *why* an agent does what it does. In other words, what is the agent's objective, which led them to act in a certain way? This is a much harder question, but also much more rewarding when answered correctly: knowing the agent's motivation allows us not only to imitate but also to understand or even influence the agent's behavior.

For this purpose, we ask the question: what is the agent optimizing for when making decisions? For instance, imagine a consumer agent that buys a certain set of products given their budget. To model the consumer's behavior, we interpret their action as buying the products with maximum utility, given a limited budget. Thus, learning how the consumer evaluates each product (that is, the utility of each product for the consumer) would allow us to understand, replicate, and possibly influence their behavior. Mathematically, we model the decision process of the agent as an optimization program, and we use Inverse Optimization (IO) as a tool to "reverse engineer" the agent's optimization program from observed behavior.

This thesis can be divided into two parts. First, we dive into the mathematical formalization of the IO problem. We look at the geometry of the mathematical objects emerging from IO problems, and we discuss what it means to solve the IO problem and different ways to do it, proposing tractable reformulations and efficient algorithms. In the second part of this thesis, we develop a tailored IO methodology to solve IO problems emerging from routing problems. We test the potential of our methodology for modeling human driving behavior on real-world problems using data from the Amazon Last Mile Routing Research Challenge. We achieve excellent results, showcasing the potential of IO to solve real-world problems. Additionally, we also developed *InvOpt*, an open-source Python package to solve general IO problems.

SAMENVATTING

Dit proefschrift gaat over het fundamentele probleem van het leren van het gedrag van besluitvormende agenten door alleen waarnemingen te doen van hoe ze zich gedragen in verschillende situaties. Als mensen doen we dit de hele tijd, en we doen het al sinds onze geboorte: denk maar aan hoe een kind leert praten en lopen. Dit proefschrift richt zich echter niet alleen op imiteren. We gaan een stap verder en proberen te leren *waarom* een agent doet wat hij doet. Met andere woorden, wat is het doel van de agent, waardoor hij op een bepaalde manier handelt? Dit is een veel moeilijkere vraag, maar ook veel belonender als deze correct wordt beantwoord: als we de motivatie van de agent kennen, kunnen we niet alleen imiteren, maar ook het gedrag van de agent begrijpen of zelfs beïnvloeden.

Hiervoor stellen we de vraag: waar optimaliseert de agent voor als hij beslissingen neemt? Stel je bijvoorbeeld een consumentenagent voor die een bepaalde set producten koopt gegeven zijn budget. Om het gedrag van de consument te modelleren, interpreteren we hun actie als het kopen van de producten met maximaal nut, gegeven een beperkt budget. Als we dus leren hoe de consument elk product beoordeelt (dat wil zeggen het nut van elk product voor de consument), kunnen we hun gedrag begrijpen, repliceren en mogelijk beïnvloeden. Wiskundig gezien modelleren we het beslissingsproces van de agent als een optimalisatieprogramma en we gebruiken *Inverse Optimization* (IO) als een hulpmiddel om het optimalisatieprogramma van de agent te "*reverse engineer*" op basis van geobserveerd gedrag.

Dit proefschrift kan in twee delen worden verdeeld. Eerst duiken we in de wiskundige formalisering van het IO-probleem. We kijken naar de geometrie van de wiskundige objecten die voortkomen uit IO-problemen, en we bespreken wat het betekent om het IO-probleem op te lossen en verschillende manieren om dit te doen, waarbij we hanteerbare herformuleringen en efficiënte algoritmen voorstellen. In het tweede deel van dit proefschrift ontwikkelen we een op maat gemaakte IO-methodologie om IO-problemen op te lossen die voortkomen uit routeringsproblemen. We testen het potentieel van onze methodologie voor het modelleren van menselijk rijgedrag op basis van echte problemen met behulp van gegevens van de Amazon Last Mile Routing Research Challenge. We behalen uitstekende resultaten en laten het potentieel van IO zien om echte problemen op te lossen. Daarnaast hebben we ook *InvOpt* ontwikkeld, een open-source Python-pakket om algemene IO-problemen op te lossen.

1

INTRODUCTION

In Inverse Optimization (IO) problems, our goal is to model the behavior of an expert agent, which given an exogenous signal, returns a response action. The underlying assumption of IO is that to compute its response, the expert agent solves an optimization problem parametric in the exogenous signal. We assume to know the constraints imposed on the expert, but not its cost function. Therefore, our goal is to model the cost function being optimized by the expert, using examples of exogenous signals and corresponding expert response actions. For example, consider the problem of learning how to route vehicles using examples from experienced drivers. Given a set of customer demands (exogenous signal), the experienced driver chooses a certain vehicle route to serve the customers (expert response). Assuming the drivers solve some kind of route optimization problem to compute their routes, where their cost function depends on the drivers' evaluation of how costly it is to drive from one customer to another, one could look at this problem as an IO problem. Thus, one could use IO tools to learn the cost function being used by these drivers, i.e., we learn the preferences of the expert drivers when driving from a certain customer to another. IO tools have been used in many application areas, for instance, the papers [BT92; FSS03; BPS17] use IO to learn the cost matrix of shortest path problems. The paper [CD12] investigates IO for the Traveling Salesperson Problem (TSP), where they study the problem of, given an edge-weighted complete graph, a single TSP tour, and a TSP solving algorithm, finding a new set of edge weights so that the given tour can be an optimal solution for the algorithm, and is closest to the original weights. Moreover, cutting-plane methods have been proposed to solve general IO for mixed-integer programs [Wan09; DW11; BCZ22], in particular, the authors of [BCZ22] propose the use of *trust regions* to lower the computational cost of generating cuts. IO has also been used to learn household activity patterns [CR12], for network learning [CRJ14; Xu+18], for learning complex model predictive control schemes [AKE21], healthcare problems [Cha+14; Cha+22], modeling of consumer behavior [BGP15], portfolio selection [BGP12], and forecast of electricity prices [SM17]. For more examples of applications of IO, we refer the reader to the recent review paper [CMZ23] and references therein.

1

The literature on IO can be roughly divided into so-called *classical* and *data-driven* IO. In classical IO, the goal is usually to find a cost function that renders a *single* signal-response pair optimal, that is, a cost function under which the observed response is optimal. A direct approach to modeling this scenario leads to a bi-level optimization problem, thus, much of the early IO literature focuses on reformulating this problem into a single-level tractable program [AO01]. This idea has been extended to different classes of optimization problems, such as conic and integer programs [Heu04; IK05; AG05; Wan09; Sch09]. On the other hand, data-driven IO usually deals with problems with multiple signal-response examples, and it is not necessarily assumed that there exists a cost function consistent with all signal-response data. In this scenario, the IO problem can be viewed as a supervised learning problem with multivariate output where the IO model forms a hypothesis class. In this view, one minimizes a training loss function to find a good "fit" to the input (response), output (optimizer) data. With this in mind, much of the data-driven IO literature focuses on the choice of the training loss function, particularly because the usual supervised learning losses lead to nonconvex programs [ASS18]. Examples of such losses are the *KKT loss* [KWB11], *first-order loss* [BGP15], *predictability loss* [ASS18], and the *suboptimality loss* [Moh+18a]. It is worth noting that in the standard IO setting, directly learning the cost function (i.e., regression with signal-pair as the input and the cost function value as the scaler output) is typically not an option since the available data contains only the signal-response pair and not the value of the unknown cost function. Recently, in a framework called “predict, then optimize”, [EG22] considers this additional information, and proposes another loss function coined as SPO. Other concepts have also been investigated in the context of IO problems, such as goodness-of-fit [CLT19], robustness against misspecification [Gho+18], and learning constraints instead of cost functions [GM21].

Notation. The trace, range space, and Moore-Penrose inverse of a matrix $Q \in \mathbb{R}^{m\times m}$ are denoted as $\mathrm{Tr}(Q)$, $\mathscr{R}(Q)$ and $Q^\dagger$, respectively. For two symmetric matrices $Q, R \in \mathbb{R}^{m\times m}$, $Q \succcurlyeq R$ means $Q - R$ is positive semidefinite. The Euclidean inner product between two vectors $x, y \in \mathbb{R}^m$ is denoted by $\langle x, y\rangle$. The set of vectors with nonnegative components is denoted as $\mathbb{R}^n_+ := \{x \in \mathbb{R}^n : x \geq 0\}$. The cardinality, complement, interior, and convex hull of a set A are denoted as $|A|$, $\overline{A}$, $\mathrm{int}(A)$ and $\mathrm{conv}(A)$, respectively. The set of integers from 1 to N is denoted as $[N]$. A set of indexed values is compactly denoted by $\{x^{[i]}\}_{i=1}^N := \{x^{[1]}, \ldots, x^{[N]}\}$. The euclidean projection of x onto a set A is defined as $\Pi_A(x) := \mathrm{argmin}_{y\in A} \|y - x\|_2$. The euclidean angle between two nonzero vectors θ and $\tilde{\theta}$ is defined as $a(\theta, \tilde{\theta}) := \arccos\langle\theta, \tilde{\theta}\rangle / (\|\theta\|_2 \|\tilde{\theta}\|_2)$. For vectors $x, y \in \mathbb{R}^m$, $x \odot y$, $\max(x, y)$ and $\exp(x)$ mean element-wise multiplication, element-wise maximum, and element-wise exponentiation, respectively. Moreover, whenever the arguments of $\exp(\cdot)$ or $\log(\cdot)$ are matrices, they should be interpreted as the usual matrix exponentiation and matrix logarithm, respectively. The vector $\mathrm{vec}(Q) \in \mathbb{R}^{pq}$ denotes the vectorization of the matrix $Q \in \mathbb{R}^{p\times q}$ formed by stacking the columns of Q into a single column vector. The symbol $\otimes$ denotes the Kronecker product. The vector $e_j \in \mathbb{R}^n$ denotes the vector of zeros except for the j'th element which is equal to one. The vector of ones is represented by $\mathbb{1}$.

1.1. PROBLEM DESCRIPTION

Let us begin by formalizing the IO problem. Let $s \in \mathbb{S}$ be an exogenous signal, and $\mathbb{S}$ be the signal space. The expert agent is assumed to solve the parametric optimization problem

$$\min_{x \in \mathbb{X}(s)} F(s,x), \tag{1.1}$$

where $\mathbb{X}(s)$ is the feasible set, and $F : \mathbb{S} \times \mathbb{X} \to \mathbb{R}$ is the expert's cost function, where we define $\mathbb{X} := \bigcup_{s \in \mathbb{S}} \mathbb{X}(s)$. In our IO formulation, the signal space $\mathbb{S}$ may contain any information that the expert uses to solve the optimization problem (1.1). For example, in the context of routing problems, the signal may contain the demands of customers, time windows for the service of customers, the set of customers that need to be served, time of the day, day of the week, weather information, etc. Since it would not be practical to formally (i.e., mathematically) define a signal space that contains all possible types of signals, we leave it as a general signal space $\mathbb{S}$. Recall that the cost F is unknown to us, and we only have access to a dataset of N pairs of exogenous signals and respective expert optimal decisions $\widehat{\mathscr{D}} := \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^{N}$, that is,

$$\hat{x}^{[i]} \in \operatorname*{argmin}_{x \in \mathbb{X}(\hat{s}^{[i]})} F(\hat{s}^{[i]}, x), \quad \forall i \in [N].$$

In this work, we always assume knowledge of the constraint set $\mathbb{X}$. For more information on IO problems with an unknown constraint set, we refer the reader to the recent survey paper [CMZ23]. Moreover, we generally assume that the observed data is feasible, that is, $\hat{x}^{[i]} \in \mathbb{X}(\hat{s}^{[i]})\ \forall i \in [N]$. For ways to handle infeasible data, see Remark 2.9, or Step 1 from Section 3.3.2 for an ad-hoc approach. We use the notation "$\hat{\cdot}$" to indicate signal-response data (e.g., $\hat{s}$ and $\hat{x}$). When using a dataset of signal-response data, we use the superscript "$[i]$" to refer to the i'th pair of the dataset, e.g., $\hat{s}^{[i]}$ and $\hat{x}^{[i]}$. Using this data, our goal is to learn a cost function that when minimized, reproduces the expert's decisions as well as possible.

In this work, we consider the following *hypothesis space* (i.e., cost function space), parametrized by θ:

$$\mathscr{F}_\phi := \left\{ \langle \theta, \phi(\cdot,\cdot) \rangle : \theta \in \mathbb{R}^p \right\}, \tag{1.2}$$

where $\theta \in \mathbb{R}^p$ is called the *cost vector* and $\phi : \mathbb{S} \times \mathbb{X} \to \mathbb{R}^p$ is called *feature mapping*, which maps a signal-response pair (s,x) to a feature vector $\phi(s,x) \in \mathbb{R}^p$. In practice, choosing a suitable mapping ϕ is part of the modeling of the IO problem. To exemplify the generality of this hypothesis space, the next example shows how to model a quadratic function as a member of $\mathscr{F}_\phi$. Other utility functions from the literature, for instance, the CES function and the Cobb-Douglas function can also be easily fitted into our framework [CK20, Appendix A].

Example 1.1 (Quadratic hypothesis)**.** *Consider a quadratic function*

$$F(s,x) = \langle x, Q_{xx} x \rangle + \langle x, Q_{xs} s \rangle + \langle x, q \rangle,$$

parametrized by $(Q_{xx}, Q_{xs}, q) \in \mathbb{R}^{n \times n} \times \mathbb{R}^{n \times m} \times \mathbb{R}^n$*. Using the identity* $\operatorname{vec}(AXB) = \langle \operatorname{vec}(X), B \otimes A^\top \rangle$*, we can rewrite this quadratic function as*

$$F(s,x) = \langle \operatorname{vec}(Q_{xx}), x \otimes x \rangle + \langle \operatorname{vec}(Q_{xs}), s \otimes x \rangle + \langle x, q \rangle = \langle \theta, \phi(s,x) \rangle,$$

where

$$\theta := \begin{bmatrix} \mathrm{vec}(Q_{xx}) \\ \mathrm{vec}(Q_{xs}) \\ q \end{bmatrix} \quad \textit{and} \quad \phi(s,x) := \begin{bmatrix} x \otimes x \\ s \otimes x \\ x \end{bmatrix}.$$

Thus, our IO goal is to find a cost vector θ such that when solving the optimization problem

$$\min_{x \in \mathbb{X}(s)} \langle \theta, \phi(s,x) \rangle, \tag{1.3}$$

we can reproduce (or approximate as well as possible) the action the expert would have taken, given the same signal s. An important object in IO problems is $\mathbb{C}$, the set of cost vectors consistent with the dataset $\widehat{\mathscr{D}}$.

Definition 1.2 (Consistent cost vectors)**.** *Given a dataset* $\widehat{\mathscr{D}} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^{N}$*, feature mapping* ϕ *and feasible set* $\mathbb{X}$*, the* set of consistent cost vectors *is defined as*

$$\mathbb{C} := \left\{ \theta \in \mathbb{R}^p : \langle \theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]}) - \phi(\hat{s}^{[i]}, x^{[i]}) \rangle \leq 0,\ \forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]}),\ \forall i \in [N] \right\}. \tag{1.4}$$

In other words, $\mathbb{C}$ corresponds to the set of cost vectors θ that render the expert responses $\hat{x}^{[i]}$'s optimal, i.e., $\hat{x}^{[i]} \in \operatorname{argmin}_{x^{[i]} \in \mathbb{X}(\hat{s}^{[i]})} \langle \theta, \phi(\hat{s}^{[i]}, x^{[i]}) \rangle$, for all $i \in [N]$. Geometrically, the set $\mathbb{C}$ is a cone, defined by the intersection of half-spaces defined by hyperplanes that pass through the origin.

1.2. OUTLINE AND CONTRIBUTIONS

In Chapter 2, we present contributions concerning novel interpretations for IO problems, tractable reformulations, loss functions, and first-order optimization algorithms. This chapter is based on P. Zattoni Scroccaro, B. Atasoy, and P. Mohajerin Esfahani, "**Learning in Inverse Optimization: Incenter Cost, Augmented Suboptimality Loss, and Algorithms**", to appear as a full paper in *Operations Research*, 2024 [ZAM24]. The main contributions of Chapter 2 are summarized as follows:

(i) **Geometric & robustness interpretations.** Motivated by the geometry of the set of consistent cost vectors, we introduce the concept of an incenter (Definition 2.3), and provide insights into its geometry (Figure 2.1) and robustness (Remark 2.4) in comparison with the *circumcenter* concept from [BFL23].

(ii) **Convex reformulations & tractability.** We develop tractable convex reformulations of the incenter (Theorem 2.5, Corollary 2.6), along with a geometric interpretation of these characterizations (Figure 2.2). This may be of particular interest as the corresponding circumcenter concept is equivalent to an intractable optimization program (Theorem 2.2). To establish this intractability result, we draw connections between the circumcenter/incenter concepts and the well-known problem of extremal volume balls (Remark 2.7). Moreover, since the problem of extremal volume balls is a special case of the problem of extremal volume ellipsoids, this connection allows us to derive "ellipsoidal" versions of the incenter and circumcenter concepts (e.g., Eq. (2.8)), which perform well in our numerical experiments (Section 2.4.1).

(iii) **Generalization to inconsistent data & Augmented Suboptimality Loss.** Generalizing to the inconsistent data setting, we propose a novel loss function for IO problems, which we name *Augmented Suboptimality Loss* (ASL) (Definition 2.8). This loss can be interpreted as a *relaxation* of the incenter concept, to handle IO problems with inconsistent data (Eq. (2.11)). In special cases, this formulation of the IO problem can be shown to be equivalent to the so-called Structured SVM formulation of structured prediction problems, revealing a connection between IO and structure prediction (Remark 2.10). We further propose a general convex reformulation of the ASL for IO problems with mixed-integer feasible sets (Theorem 2.12 and Corollary 2.13), which generalizes several reformulations from the literature (Remark 2.14).

(iv) **Tailored first-order algorithm: Stochastic Approximate Mirror Descent.** Motivated by the structure of IO loss functions, we propose a novel first-order algorithm, which we name *Stochastic Approximate Mirror Descent* (SAMD). This algorithm exploits the finite sum structure of the IO problem to compute stochastic gradients, uses approximate evaluating IO loss functions to compute approximate subgradients, and uses mirror descent update steps for problems with favorable geometry (Sections 2.3.1, 2.3.2, and 2.3.3). We prove convergence rates for this algorithm (Proposition 2.16) and show how the components of this algorithm can be tailored to exploit the structure of IO loss functions (Algorithm 1).

In Chapter 3, we tailor and extend the general IO methodology introduced in Chapter 2 for the case of routing problems. Our tailored approach is flexible to what type of routing problem the expert is assumed to solve, handles cases when there is a large number of routing examples, as well as cases when solving the routing problem is computationally expensive. This chapter is based on P. Zattoni Scroccaro, P. van Beek, P. Mohajerin Esfahani, and B. Atasoy, "**Inverse Optimization for Routing Problems**", *Transportation Science*, 2024 [Zat+24]. The main contributions of Chapter 3 are summarized as follows:

(i) **IO methodology for routing problems.** We propose an IO methodology, which has the following specifications tailored for routing problems:

  (a) *Hypothesis class*: We introduce a hypothesis class of affine cost functions with nonnegative cost vectors representing the weights of edges of a graph, along with an affine term that can capture extra desired properties for the model (Section 3.1.1).

  (b) *Loss function*: We introduce a loss function for IO applied to routing problems (Section 3.1.2). This loss function extends the Augmented Suboptimality Loss by using our affine hypothesis class. Moreover, exploiting the fact that the decision variables of the routing problem can be modeled using binary variables, we show that the nonconvex cost function of the inner minimization problem of the loss function can be equivalently reformulated as a convex function (Proposition 3.1). This result is of independent interest since this reformulation can be used for any IO problem with binary decision variables.

(c) *First-order algorithm*: We also design a first-order algorithm specialized to minimize our tailored IO loss function, and is particularly efficient for IO problems with large datasets and with computationally expensive decision problems, e.g., large VRPs (Section 3.1.3). Compared to the SAMD algorithm, our algorithm is tailored to our affine hypothesis function and new loss function reformulation and also uses a "*reshuffled*" sampling strategy, which improves its empirical performance compared to the uniform sampling employed by the SAMD algorithm (Section 3.2.2).

(d) *Modeling flexibility*: We showcase the flexibility of our IO methodology by demonstrating how three specific instances of routing problems (CVRP, VRPTW, and TSP) can be modeled using our framework (Sections 3.2.1, 3.2.2, and 3.2.3). We present numerical results that give intuition on how our tailored algorithm works for routing problems (Figure 3.2), as well as its efficacy in handling large routing problems (Figure 3.3).

(ii) **Application to the Amazon Challenge.** We evaluate our IO methodology on the Amazon Challenge, namely, we learn the drivers' preferences in terms of geographical city zones using IO. We present results for a general IO approach as well as for the tailored approach developed in this chapter, showcasing how insights about the structure of the problem at hand can be seamlessly integrated into our IO methodology, illustrating its flexibility and modeling power (Sections 3.4.1). Our approach achieves a final Amazon score of **0.0302**, which ranks 2nd compared to the 48 models that qualified for the final round of the Amazon Challenge (Figure 3.9). Moreover, using an approximate TSP solver and a fraction of the training dataset, we can learn a good routing model in just a few minutes, demonstrating the possibility of using our IO approach for real-time learning problems (Table 3.2). All of our experiments are reproducible, and the underlying source code is available in [Zat23a].

IO python package. In addition to the works reported in Chapters 2 and 3, we have also developed the *InvOpt* Python package: `https://github.com/pedroszattoni/invopt` [Zat23b]. This is a general library of functions with implementations of all algorithms and formulations presented in Chapter 2 and Chapter 3, as well as other standard IO methods to solve (continuous) linear and quadratic IO problems. Moreover, all numerical experiments from Chapter 2 and part of the experiments from Chapter 3 are available in InvOpt's GitHub repository as usage examples for the package.

1.3. OTHER CONTRIBUTIONS AND CURRENT WORK

In addition to the works reported in Section 1.2, we have also worked on the following project during the PhD.

Online convex optimization with predictions. In the past few years, Online Convex Optimization (OCO) has received notable attention in the control literature thanks to its flexible real-time nature and powerful performance guarantees. In [ZKM23], we propose new step-size rules and OCO algorithms that simultaneously exploit gradient predictions, function predictions and dynamics, features particularly pertinent to control

applications. The proposed algorithms enjoy static and dynamic regret bounds in terms of the dynamics of the reference action sequence, gradient prediction error, and function prediction error, which are generalizations of known regularity measures from the literature. We present results for both convex and strongly convex costs. We validate the performance of the proposed algorithms in a trajectory tracking case study, as well as portfolio optimization using real-world datasets.

Moreover, we are currently working on two projects: kernel methods for IO and IO for dynamic problems.

Kernel methods for IO. In our current project [Lon+24], we address the issues of the choice of the feature mapping ϕ and the complexity of the IO problem w.r.t. the size of the feature vector. For some classes of IO problems, both issues can be handled using *kernel methods*, where we reformulate the IO problem in terms of a *kernel function* $\kappa(\hat{s}_1, \hat{x}_1, \hat{s}_2, \hat{x}_2) = \langle \phi(\hat{s}_1, \hat{x}_1), \phi(\hat{s}_2, \hat{x}_2) \rangle$. This way, by choosing standard kernels (e.g., polynomial or Gaussian) one does not need to design the feature mapping ϕ. Also, $\kappa(\hat{s}_1, \hat{x}_1, \hat{s}_2, \hat{x}_2) \in \mathbb{R}$, which is independent of the size of $\phi(\hat{s}_1, \hat{x}_1)$, so the complexity of the IO problem (in other words, the number of parameters to be optimized) does not depend anymore on the size of the feature vector. On the other hand, the complexity of the resulting IO reformulation will depend on the size of the training dataset, which for large-scale applications, requires specialized optimization algorithms, for instance, coordinated descent methods.

IO for dynamical decision-making problems. In dynamical decision-making problems, decisions must be taken sequentially, and the signal observed at time $t+1$ may be a function of the decision taken at time t. In other words, we can interpret the dataset $\widehat{\mathscr{D}} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^{N}$ as coming from a dynamical system, where $\hat{s}^{[t+1]} = f(\hat{s}^{[t]}, \hat{x}^{[t]})$, for some function f. For instance, the signal $\hat{s}^{[t+1]}$ could represent the state of the dynamical system and $\hat{x}^{[t+1]}$ the control input from an expert controller, where f is the dynamics mapping of the system. For such problems, ignoring their dynamical nature can lead to poor out-of-sample performance of the learned IO model, since small errors can be compounded by the dynamical aspect of the problem. Thus, the goal of this project is to develop IO methodologies tailored to dynamical decision-making problems, with a particular interest in applications to dynamic vehicle routing problems, and possibly combine ideas from the *Reinforcement Learning* and *Contextual Bandits* algorithms.

2

LEARNING IN INVERSE OPTIMIZATION

Recently, [BFL23] shows that given a dataset of expert examples and the set $\mathbb{C}$, which is defined as the set of cost vectors consistent with the dataset, the optimal solution to the IO problem, for a certain regret performance measure, is the so-called *circumcenter* of the set $\mathbb{C}$. To the best of our knowledge, this is the first minimax regret result for IO problems. To derive this result, the authors exploit the geometry of IO problems and provide strong regret guarantees. The general optimization program associated with computing the circumcenter vector, however, turns out to contain intractable problem instances. Moreover, when the IO dataset is inconsistent with a single cost function (which is expected in real-world problems), it is not clear how one could generalize the circumcenter concept to solve the IO problem. These shortcomings motivate us to propose a new concept called “incenter" to select an IO cost vector from the set of consistent costs.

2.1. INCENTER COST VECTOR

From Definition 1.2, one can observe that a trivial element in the set $\mathbb{C}$ is $\theta = 0$. However, this trivial choice is not really of interest since it yields any response $x \in \mathbb{X}(s)$ to the signal s an optimal solution, hence no differentiation between different responses. Therefore, when $\mathbb{C}$ contains other elements than $\theta = 0$, that is, the data is consistent with the hypothesis space in the sense that the expert's true cost is $F(s,x) = \langle \theta^\star, \phi(s,x) \rangle$ for some nonzero element $\theta^\star$, it is natural to restrict the search space to $\mathbb{C} \setminus \{0\}$. It is also worth noting that a similar issue concerns the feature map ϕ. Namely, from a modeling perspective, it is important to consider a feature class where $\phi(s,x)$ is not identically zero in the second argument over the set $\mathbb{X}(s)$ for typical signal variables $s \in \mathbb{S}$. This assumption, together with excluding the trivial cost vector $\theta = 0$ from the search space, is necessary. Thus, in this section, we assume $\mathbb{C} \setminus \{0\} \neq \emptyset$ and that ϕ is not trivially zero, and we discuss different ways to choose a cost vector from the set $\mathbb{C}$. A straightforward way to do this is

This chapter is based on [ZAM24].

by solving a feasibility optimization problem, which we call *feasibility program*:

$$\begin{aligned} \min_{\theta} \quad & 0 \\ \text{s.t.} \quad & \langle \theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]}) - \phi(\hat{s}^{[i]}, x^{[i]}) \rangle \leq 0 \qquad \forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]}),\ \forall i \in [N] \\ & \|\theta\|_2 = 1. \end{aligned} \tag{2.1}$$

Problem (2.1) simply translates the idea that we wish to find some nonzero $\theta \in \mathbb{C}$. Notice that the normalization constraint $\|\theta\|_2 = 1$ excludes the trivial solution $\theta = 0$, and it is nonrestrictive in the sense that $\operatorname{argmin}_{x \in \mathbb{X}(s)} \langle \theta, \phi(s,x) \rangle = \operatorname{argmin}_{x \in \mathbb{X}(s)} \langle \alpha\theta, \phi(s,x) \rangle$, for any $\alpha > 0$. By solving the IO problem using the feasibility program (2.1), we implicitly suggest that any cost vector $\theta \in \mathbb{C} \setminus \{0\}$ is an equally good solution to the IO problem. However, one could argue that there are more principled ways to choose a cost vector from the set of consistent vectors $\mathbb{C}$.

2.1.1. GEOMETRY, ROBUSTNESS, AND TRACTABILITY

Considering two cost vectors θ and $\theta^\star$ with their corresponding responses defined as

$$x^\theta \in \operatorname*{argmin}_{x \in \mathbb{X}(\hat{s})} \langle \theta, \phi(\hat{s}, x) \rangle \quad \text{and} \quad x^\star \in \operatorname*{argmin}_{x \in \mathbb{X}(\hat{s})} \langle \theta^\star, \phi(\hat{s}, x) \rangle, \tag{2.2a}$$

the authors in [BFL23] define the regret

$$R(\theta, \theta^\star) := \langle \theta^\star, \phi(\hat{s}, x^\theta) - \phi(\hat{s}, x^\star) \rangle. \tag{2.2b}$$

The regret (2.2b) is a distance function between two cost vectors that measures the difference via the impact of their respective optimizers x^θ and $x^\star$ evaluated through the second argument, here the ground truth $\theta^\star$. As shown in [BFL23], the worst-case optimal cost vector θ for the regret $R(\theta, \theta^\star)$ is the so-called *circumcenter* of $\mathbb{C}$.

Definition 2.1 (Circumcenter [BFL23])**.** *Let $\mathbb{C}$ be a nonempty set. A* circumcenter *of $\mathbb{C}$ is defined as*

$$\theta^{\mathrm{c}} \in \operatorname*{argmin}_{\theta \neq 0} \max_{\substack{\tilde{\theta} \in \mathbb{C} \\ \tilde{\theta} \neq 0}} a(\theta, \tilde{\theta}). \tag{2.3}$$

In the original definition of [BFL23], the non-zero constraints over the cost vectors are enforced via $\|\theta\|_2 = \|\tilde{\theta}\|_2 = 1$, which does not change the circumcenter since the angle in the objective is scale-invariant (recall that the euclidean angle between two nonzero vectors θ and $\tilde{\theta}$ is defined as $a(\theta, \tilde{\theta}) = \arccos \langle \theta, \tilde{\theta} \rangle / (\|\theta\|_2 \|\tilde{\theta}\|_2)$). Geometrically, θ^{c} can be interpreted as a point in the axis of the revolution cone with the smallest aperture angle containing $\mathbb{C}$. Informally speaking, the circumcenter approach chooses a cost vector by looking for the vector in $\mathbb{C}$ furthest away (in terms of the angle) from the "corners" of $\mathbb{C}$ (more precisely, from the extreme rays of $\mathbb{C}$). However, in this study we identify the following three possible shortcomings when using the circumcenter concept to choose an IO model:

(i) **Computational tractability:** Computing the circumcenter, in general, turns out to be intractable.

(ii) **Robustness interpretation:** While the circumcenter is, by definition, robust to an adversarial (i.e., worst-case) ground truth data-generating model, the challenge in practice is often related to noisy data.

(iii) **Inconsistent data:** The circumcenter is only defined for problems with consistent data (i.e., nonempty $\mathbb{C}$) and, to the best of our knowledge, cannot be straightforwardly extended to IO problems with inconsistent data.

In the rest of this section, we first formalize the tractability result (i), and then introduce a new notion that addresses these possible shortcomings related to circumcenter.

Theorem 2.2 (Intractability of circumcenter)**.** *Assume $\mathbb{C}$ is a nonempty polyhedral cone. Then, solving the inner maximization of (2.3) for any θ is equivalent to maximizing a quadratic function over a polytope, which is NP-hard.*

Proof. Section 2.C.1. □

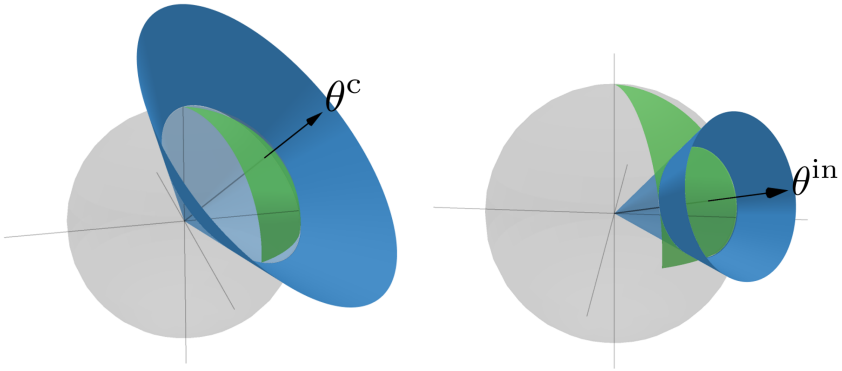
Definition 2.3 (Incenter)**.** *Let $\mathbb{C}$ be a nonempty set. An* incenter *of $\mathbb{C}$ is defined as*

$$\theta^{\text{in}} \in \operatorname*{argmax}_{\theta \neq 0} \min_{\substack{\tilde{\theta} \in \overline{\text{int}(\mathbb{C})} \\ \tilde{\theta} \neq 0}} a(\theta, \tilde{\theta}). \tag{2.4}$$

We will show how the incenter problem (2.4) can be reformulated as a tractable convex optimization problem (as opposed to the NP-hardness of the circumcenter), and by relaxing the reformulation, it can be straightforwardly extended to handle problems with inconsistent data. Let us first elaborate more on the intuition behind the incenter vector in Definition 2.3 compared to the circumcenter vector in Definition 2.1. Geometrically, an incenter of $\mathbb{C}$ can be interpreted as a vector furthest away (in terms of the angle) from the *exterior* of $\mathbb{C}$, or in other words, the vector in $\mathbb{C}$ furthers away from the boundary of $\mathbb{C}$. To provide insight into these different centers, we visualize them in a simple three-dimensional space in Figure 2.1. Interestingly, these notions also have robustness interpretations, which are different from existing learning models in which the robustness is introduced explicitly via regularization or distributional robust approaches [Moh+18a].

Remark 2.4 (Robustness: circumcenter vs. incenter)**.** *A circumcenter θ^{c} in (2.3) can be interpreted as the vector most robust to an adversary with the authority to choose the true data-generating cost vector in $\mathbb{C}$ to be furthest away from our learned model, which is a notion captured by the regret performance measure introduced in [BFL23]. On the other hand, the incenter θ^{in} (2.4) can be viewed as the vector furthest away from the boundary of $\mathbb{C}$. Since each facet of $\mathbb{C}$ is determined by a signal-response $(\hat{s}^{[i]}, \hat{x}^{[i]})$ pair in the dataset of the IO problem, the incenter vector can be interpreted as the cost vector most robust to perturbations of the training dataset (i.e., perturbations of the facets of $\mathbb{C}$).*

Let us provide some additional details supporting the robustness interpretation in Remark 2.4. To formalize this robustness notion, recall the definition of the set of consistent cost vectors $\mathbb{C} = \{\theta \in \mathbb{R}^p : \langle \theta, \Delta(\hat{s}^{[i]}, \hat{x}^{[i]}, x) \rangle \leq 0,\ \forall x \in \mathbb{X}(\hat{s}^{[i]}),\ \forall i \in [N]\}$, where we define $\Delta(\hat{s}^{[i]}, \hat{x}^{[i]}, x) := (\phi(\hat{s}^{[i]}, \hat{x}^{[i]}) - \phi(\hat{s}^{[i]}, x^{[i]}))/\|\phi(\hat{s}^{[i]}, \hat{x}^{[i]}) - \phi(\hat{s}^{[i]}, x^{[i]})\|_2$ if $\phi(\hat{s}^{[i]}, \hat{x}^{[i]}) \neq \phi(\hat{s}^{[i]}, x^{[i]})$, and $\Delta(\hat{s}^{[i]}, \hat{x}^{[i]}, x) := 0$ otherwise. Differently from the definition in Eq. (1.4),



(a) Circumcenter θ^{c} defined in (2.3).

(b) Incenter θ^{in} defined in (2.4).

Figure 2.1: Geometrical visualization of the circumcenter and incenter vectors. The green regions are the intersection of $\mathbb{C}$ with the sphere and the blue cones are revolution cones with an aperture angle equal to the optimal value of (2.3) and (2.4).

here we simply normalize the vectors $\phi(\hat{s}^{[i]},\hat{x}^{[i]})-\phi(\hat{s}^{[i]},x^{[i]})$, which does not change the set $\mathbb{C}$. Notice that the vectors $\Delta(\hat{s}^{[i]},\hat{x}^{[i]},x)$ are defined by the data $\{(\hat{s}^{[i]},\hat{x}^{[i]})\}_{i=1}^{N}$, and in turn define the set $\mathbb{C}$. Thus, the vector $\theta\in\mathbb{C}$ most robust to perturbations in the data, i.e., perturbation of $\Delta(\hat{s}^{[i]},\hat{x}^{[i]},x)$, can be found by solving the minimax problem

$$\begin{array}{lll}\max\limits_{\substack{\|\theta\|_2=1\\ \theta\in\mathbb{C}}} & \min\limits_{\substack{w\in\mathbb{R}^p\\ (\hat{s}^{[k]},\hat{x}^{[k]})\in\{(\hat{s}^{[i]},\hat{x}^{[i]})\}_{i=1}^{N}\\ x\in\mathbb{X}(\hat{s}^{[k]})}} & \|w\|_2^2\\ & \text{s.t.} & \langle\theta,\Delta(\hat{s}^{[k]},\hat{x}^{[k]},x)+w\rangle=0.\end{array}\tag{2.5}$$

In problem (2.5), the "max player" optimizes for the vector $\theta\in\mathbb{C}$ that requires the largest perturbation w so that it lies in the hyperplane $\langle\theta,\Delta(\hat{s}^{[k]},\hat{x}^{[k]},x)+w\rangle=0$ (i.e., a perturbed facet of $\mathbb{C}$). The "min player" tries to find the "most vulnerable" $\Delta(\hat{s}^{[k]},\hat{x}^{[k]},x)$, that is, the facet of $\mathbb{C}$ that requires the smallest perturbation vector w to make $\langle\theta,\Delta(\hat{s}^{[k]},\hat{x}^{[k]},x)+w\rangle=0$. It can be shown that the optimal perturbation is $w=-\big(\langle\theta,\Delta(\hat{s}^{[k]},\hat{x}^{[k]},x)\rangle/\|\theta\|_2^2\big)\theta$, and substituting it in the objective function of (2.5), one can show that the optimization problem (2.5) is equivalent to the incenter reformulation of Theorem 2.5.

As a final comment, even ignoring the computational cost of computing the circumcenter vector versus computing the incenter vector, for cases when the set of consistent vectors $\mathbb{C}$ is not empty, the incenter approach may perform better in practice (e.g., see the results in Section 2.4.1). In light of the discussion in Remark 2.4, this suggests that approaches robust to perturbations in the data (i.e., incenter) might be preferred to the one that is robust to an adversarial true data-generating model (i.e., circumcenter).

2.1.2. REFORMULATIONS

Next, we present a reformulation of (2.4). This reformulation will be used to derive tractable incenter-based approaches to IO problems, and we show that if $\text{int}(\mathbb{C})\neq\emptyset$, one

can compute the incenter vector by solving an optimization problem without a norm equality constraint.

Theorem 2.5 (Incenter reformulation)**.** *Problem* (2.4) *can be reformulated as*

$$\begin{aligned}(\theta^{\text{in}}, r^{\text{in}}) \in \arg\max_{\theta, r}\ & r \\ \text{s.t. } & \langle \theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]}) - \phi(\hat{s}^{[i]}, x^{[i]})\rangle + r\|\phi(\hat{s}^{[i]}, x^{[i]}) - \phi(\hat{s}^{[i]}, \hat{x}^{[i]})\|_2 \le 0, \\ & \qquad \forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]}), \forall i \in [N] \\ & \|\theta\|_2 = 1.\end{aligned} \tag{2.6}$$

Proof. Section 2.C.2. □

Corollary 2.6 (Incenter convex characterization)**.** *Assume* $\text{int}(\mathbb{C}) \neq \emptyset$*. Then, problem* (2.6) *can be solved as*

$$\begin{aligned}\bar{\theta}^{\text{in}} \in \arg\min_{\theta}\ & \|\theta\|_2 \\ \text{s.t. } & \langle \theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]}) - \phi(\hat{s}^{[i]}, x^{[i]})\rangle + \|\phi(\hat{s}^{[i]}, x^{[i]}) - \phi(\hat{s}^{[i]}, \hat{x}^{[i]})\|_2 \le 0 \\ & \qquad \forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]}),\ \forall i \in [N],\end{aligned} \tag{2.7}$$

where $\theta^{\text{in}} = \bar{\theta}^{\text{in}} / \|\bar{\theta}^{\text{in}}\|_2$ *is an optimal solution of problem* (2.6)*.*

Proof. Section 2.C.3. □

Figure 2.2 presents a geometrical intuition behind the alternative reformulation of Corollary 2.6 in a simple 2D example. Intuitively, the condition $\text{int}(\mathbb{C}) \neq \emptyset$ guarantees that the optimal $r > 0$ in Theorem 2.5, and since the 1 in the norm equality constraint is arbitrary, we can rescale the problem to get rid of the optimization variable r. Thus, exploiting the nonempty interior assumption $\text{int}(\mathbb{C}) \neq \emptyset$, we were able to formulate problem (2.7) without a norm equality constraint, making it a convex optimization problem. In particular, it is tractable whenever $\mathbb{X}(\hat{s}^{[i]})$ has finitely many elements for all $i \in [N]$. In Sections 2.2.2 and 2.3, we discuss approaches to deal with cases when $\mathbb{X}(\hat{s}^{[i]})$ may have infinitely many elements. The tractability results in Theorem 2.2 and Corollary 2.6 build on a connection to well-known problems in the optimization literature: computing extremal volume balls.

Remark 2.7 (Connection with extremal volume balls)**.** *Interestingly, the circumcenter (resp. incenter) problem is closely related to the problem of finding the minimum (resp. maximum) volume ball that covers (resp. lies inside) a nonempty polyhedron. The difference is that, instead of optimizing the volume of a ball so that it covers (resp. lies inside) a nonempty polyhedron, we optimize the aperture angle of a circular cone, so that it covers (resp. lies inside) a polyhedral cone (see Figure 2.1). It is known that computing the minimum volume ball that contains a polyhedron defined by linear inequalities is an intractable convex optimization problem [Nem96, Section 10.5.1], and Theorem 2.2 indicates that the circumcenter has the same property. Moreover, finding the maximum volume ball that lies inside a polyhedron defined by linear inequalities (a.k.a. the Chebyshev center of the polyhedron) is known to be equivalent to a convex optimization problem*

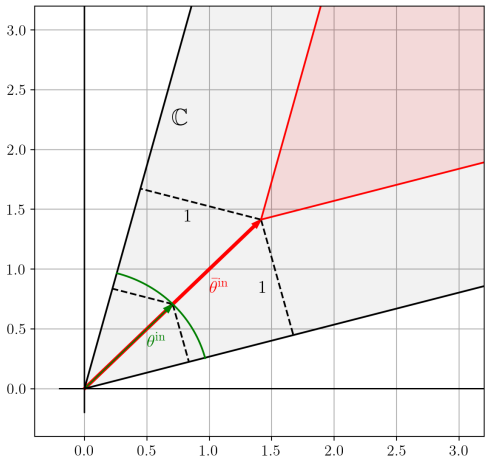


Figure 2.2: Geometrical illustration of Corollary 2.6 in a simple 2D example. The grey region represents $\mathbb{C}$, the green region represents the feasible set of (2.6), and the red region represents the feasible set of (2.7). As can be seen, the optimal solution of (2.7) can be interpreted as the smallest norm vector inside an inner cone with boundaries 1 unit away from the boundaries of $\mathbb{C}$. Also, it can be seen that by normalizing $\bar{\theta}^{\text{in}}$, we retrieve θ^{in}, an optimal solution of (2.6).

[BV04, Section 8.5.1], and Corollary 2.6 also confirms that the incenter has the same property.

Since the problem of extremal volume balls is a special case of the more general problem of extremal volume ellipsoids [BV04, Section 8.4], one natural generalization of the incenter and circumcenter concepts is to use *ellipsoidal cones*. For instance, an ellipsoidal generalization of the circumcenter concept is used by [BFL23] for online IO problems. An ellipsoidal generalization of the incenter reformulation of Theorem 2.5 is

$$\begin{aligned} \max_{\theta, A} \quad & \log(\det(A)) \\ \text{s.t.} \quad & \langle \theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]}) - \phi(\hat{s}^{[i]}, x^{[i]}) \rangle + \| A(\phi(\hat{s}^{[i]}, x^{[i]}) - \phi(\hat{s}^{[i]}, \hat{x}^{[i]})) \|_2 \leq 0, \\ & \qquad\qquad \forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]}), \forall i \in [N] \\ & \|\theta\|_2 = 1, \quad A \succcurlyeq 0, \end{aligned} \tag{2.8}$$

which is based on [BV04, Section 8.4.2]. Even though, to the best of our knowledge, there are no theoretical results for the performance of ellipsoidal incenter and circumcenter approaches for offline IO problems, their performance in our numerical experiments in Section 2.4 is indeed promising.

We end this section presenting a generalization of the program (2.7). This generalization will give us the flexibility to exploit structures specific to different IO problems.

We generalize (2.7) as

$$\begin{aligned} \min_{\theta} \quad & \mathscr{R}(\theta) \\ \text{s.t.} \quad & \langle \theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]}) - \phi(\hat{s}^{[i]}, x^{[i]}) \rangle + d(\hat{x}^{[i]}, x^{[i]}) \leq 0, \qquad \forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]}),\ \forall i \in [N] \\ & \theta \in \Theta. \end{aligned} \tag{2.9}$$

The generalization occurs on three levels:

(i) **Regularizer**: We generalize the objective function to a general regularization function $\mathscr{R} : \mathbb{R}^p \to \mathbb{R}$. For example, we could have $\mathscr{R}(\theta) = \|\theta - \hat{\theta}\|_2^2$, where $\hat{\theta}$ is an a priory belief or estimate of the true cost vector, or $\mathscr{R}(\theta) = \|\theta\|$ for a general norm.

(ii) **Distance in the response space**: Instead of computing the distance between vectors using the Euclidean distance, we consider a general distance function $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}_+$. For instance, $d(\hat{x}, x) = \|\hat{x} - x\|$ on continuous spaces, or $d(\hat{x}, x) = I(\hat{x}, x)$ on discrete spaces, where $I(\hat{x}, x) = 0$ if $\hat{x} = x$, otherwise $I(\hat{x}, x) = 1$. We note that d could also be a function of $\hat{s}$. For instance, we could use it to penalize the distance between $\hat{x}$ and x in the *feature space*, by choosing $d(\hat{x}, x) = \|\phi(\hat{s}, \hat{x}) - \phi(\hat{s}, x)\|$. For simplicity, we omit this dependency.

(iii) **Prior information constraint**: We add the additional constraint $\theta \in \Theta$. The set Θ is used to encode any prior information or assumption we may have on the expert's true cost function, e.g., nonnegativity of the cost vector.

Notice that as long as $\mathscr{R}(\theta)$ is convex in θ and the set Θ is convex, Problem (2.9) is a convex optimization problem.

2.2. AUGMENTED SUBOPTIMALITY LOSS

In the previous section, we have shown how one can tackle IO problems when the dataset $\widehat{\mathscr{D}}$ is consistent with some cost $F(s, x) = \langle \theta^\star, \phi(s, x) \rangle \in \mathscr{F}_\phi$. Moreover, to use the proposed approaches in practice, we also need to be able to list all elements of $\mathbb{X}(\hat{s}^{[i]})$ for all $i \in [N]$, since each of these elements induces one inequality constraint (e.g., see (2.9)). In this section, we present approaches to tackle an IO problem based on a *loss function*, which will allow us to handle problems with possibly inconsistent data and when the cardinality of $\mathbb{X}(\hat{s}^{[i]})$ is very large (or even infinite). We first introduce a novel loss function for IO problems and show how solving the IO problem using this loss can be interpreted as a relaxation of the optimization program (2.9).

Recall that in IO problems, our goal is to find a parameter vector θ, such that when solving the optimization problem $\min_{x \in \mathbb{X}(s)} \langle \theta, \phi(s, x) \rangle$, we can reproduce (or approximate) the action the expert would have taken given the same signal $s \in \mathbb{S}$ (see Section 1.1). In the data-driven IO literature, this problem is usually posed as a (regularized) empirical loss minimization problem

$$\min_{\theta \in \Theta} \kappa \mathscr{R}(\theta) + \frac{1}{N} \sum_{i=1}^{N} \ell_\theta(\hat{s}^{[i]}, \hat{x}^{[i]}), \tag{2.10}$$

where $\mathscr{R} : \mathbb{R}^p \to \mathbb{R}$ is a regularization function, κ is a nonnegative regularization parameter, and $\ell_\theta : \mathbb{S} \times \mathbb{X} \to \mathbb{R}$ is some loss function, designed in a way that, by solving (2.10), we find a cost vector θ such that the function $\langle \theta, \phi(s,x) \rangle$ achieves our IO goal. Similar to supervised learning methods in machine learning, the regularization parameter κ should be chosen to improve the out-of-sample performance of the learned model, e.g., using cross-validation techniques. In this chapter, we propose a new loss function for IO problems, which we name *Augmented Suboptimality Loss* (ASL).

Definition 2.8 (Augmented Suboptimality Loss)**.** *Given a signal-response pair* $(\hat{s}, \hat{x})$*, the Augmented Suboptimality Loss of a cost vector* θ *is*

$$\ell_\theta(\hat{s}, \hat{x}) := \max_{x \in \mathbb{X}(\hat{s})} \left\{ \langle \theta, \phi(\hat{s}, \hat{x}) - \phi(\hat{s}, x) \rangle + d(\hat{x}, x) \right\}, \tag{ASL}$$

where ϕ *is a feature mapping and* d *is a distance function.*

To simplify the notation, we omit the dependence of the ASL on ϕ and d. We note that the ASL can be the well-known *Suboptimality Loss* [Moh+18a] for the special case with no distance penalization, i.e., $d(\hat{x}, x) = 0$.

2.2.1. CONNECTIONS WITH INCENTER

Next, we show how to solve problem (2.10) when the loss function is the ASL. This reformulation is useful to tackle IO problems with inconsistent data, where there is no cost vector $\theta^\star$ that describes the dataset $\widehat{\mathscr{D}} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$ perfectly. Using a standard epigraph reformulation, one can see that the optimization problem (2.10) with the ASL yields the program

$$\begin{aligned} \min_{\theta, \beta_1, \ldots, \beta_N} \; & \kappa \mathscr{R}(\theta) + \frac{1}{N} \sum_{i=1}^N \beta_i \\ \text{s.t.} \; & \langle \theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]}) - \phi(\hat{s}^{[i]}, x^{[i]}) \rangle + d(\hat{x}^{[i]}, x^{[i]}) \leq \beta^{[i]}, \; \forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]}), \; \forall i \in [N] \\ & \theta \in \Theta. \end{aligned} \tag{2.11}$$

Problem (2.11) is closely related to the incenter reformulation (2.9). More precisely, (2.11) can be interpreted as a *relaxation* of (2.9) to handle IO problems with inconsistent data. To see this, simply notice that we can arrive at (2.11) by adding *slack variables* to the constraints of (2.9) and using κ as a trade-off parameter between the sum of slack variables (i.e., the total violation of the constraints of (2.9)) and the regularization term $\mathscr{R}(\theta)$. Another way to see (2.10) as a relaxation of (2.9), is to write (2.9) in a "loss minimization" form. Namely, notice that (2.9) can be written as

$$\min_{\theta \in \Theta} \mathscr{R}(\theta) + \mathbb{I}\left(\frac{1}{N} \sum_{i=1}^N \ell_\theta(\hat{s}^{[i]}, \hat{x}^{[i]}) \right), \tag{2.12}$$

where the indicator function $\mathbb{I}(a) = 0$ if $a \leq 0$, otherwise $\mathbb{I}(a) = \infty$. Thus, (2.10) can also be interpreted as Lagrangian relaxation of (2.12). In Appendix 2.A, we present a different way to motivate the ASL, namely, as a convex surrogate of the so-called *predictability loss*. There, we also discuss several properties of the ASL which are relevant to IO problems.

So far, we have only considered the case when the data $\{\hat{x}^{[i]}\}_{i=1}^N$ is feasible, i.e., $\hat{x}^{[i]} \in \mathbb{X}(\hat{s}^{[i]})$. However, in practice, it may be the case that $\hat{x}^{[i]} \notin \mathbb{X}(\hat{s}^{[i]})$ for some $i \in [N]$. This may happen due to noise in the data acquisition, or simply due to a mismatch in the choice of the set mapping $\mathbb{X}$ compared with the true constraint set used by the expert agent. For example, consider the problem where the signal $\hat{s}^{[i]}$ represents a graph, and $\mathbb{X}(\hat{s}^{[i]})$ is the set of *Hamiltonian cycles* over this graph (i.e., a graph cycle that visits each node exactly once). This scenario is common when modeling vehicle routing problems, where the nodes of the graph represent a set of customers, and the Hamiltonian cycle represents the sequence of customers a driver visited (for example, to deliver packages). In this scenario, infeasible data could reflect, for example, missing data on the complete cycle driven by the driver (i.e., noise in the data acquisition), or the fact that in reality, the driver can visit the same customer multiple times to deliver missing packages (i.e., mismatch in the choice of the set mapping $\mathbb{X}(\hat{s}^{[i]})$).

Remark 2.9 (Handling infeasible data)**.** *One way to handle infeasible data is just trying to ignore these cases, treating them as outliers in the data. Numerically, since the nonnegativity of the ASL depends on the assumption that $\hat{x}^{[i]} \in \mathbb{X}(\hat{s}^{[i]})\ \forall i \in [N]$ (see Proposition 2.21), having infeasible data may lead to negative loss values. Thus, a possible way to handle these cases is to use a modified loss function $\tilde{\ell}_\theta(\hat{s}, \hat{x}) = \max\{0, \ell_\theta(\hat{s}, \hat{x})\}$, which simply ignores negative loss values (i.e., infeasible data). This technique shares the same principle as the bounded rationality loss of [Moh+18a]. Importantly, this modification preserves the convexity of the loss minimization problem. For instance, this modification is equivalent to simply adding the constraints $\beta_i \geq 0,\ \forall i \in [N]$ to (2.11).*

The IO problem shares some similarities with *structured prediction* problems, which commonly refer to the class of supervised learning problems, where the output space consists of a finite set of structured objects (e.g., graphs), instead of a simple set of labels, like in standard multiclass classification problems [Tas+05]. Interestingly, the IO formulation in Eq. (2.11) is closely related to some formulations proposed to solve structured prediction problems.

Remark 2.10 (Connections with structured prediction)**.** *In the incenter program (2.11), when $\mathscr{R}$ is the Euclidean norm, d is the Hamming distance, and feature function ϕ is linear in x, learning the incenter cost vector is equivalent to the Maximum Margin Planning problem presented in [RBZ06], which is a type of structured prediction problem. More generally, formulation (2.11) is closely related to the so-called Structured Support Vector Machine (SSVM) approach for structured prediction problems [Tso+05; NL+11].*

The connection between IO problems with discrete feasible sets and the SSVM approach to structured prediction problems is related to the fact that when the constraint set $\mathbb{X}(\hat{s})$ has finitely many elements, each of these elements can be interpreted as a class in a multiclass classification problem. Under this interpretation, the optimization problem (2.11) can be viewed as a generalization of the soft-margin SVM problem, where the "classes" $x \in \mathbb{X}(\hat{s})$ can be complex structured objects, which is precisely the SSVM problem. Moreover, the concept of "margin" in the SVM framework is related to the distance function d in (2.11), which comes from the fact that in our incenter-based IO formulation, we want to maximize the angle between the incenter vector and the boundaries of

the set $\mathbb{C}$ (see Figure 2.1b). These observations reveal an interesting connection between incenter-based approaches for IO problems with discrete constraint sets and SSVM approaches for structured prediction problems.

In the last part of this section, we revisit this relation between the incenter and circumcenter concepts through the lens of the regret R defined in (2.2). In fact, given two cost vectors $(\theta, \theta^\star)$, there is an asymmetry $R(\theta,\theta^\star) \neq R(\theta^\star,\theta)$, interestingly, each of which relating to one of these concepts.

Remark 2.11 (Connections with regret)**.** *Consider the cost vectors $(\theta,\theta^\star)$ and the regret (2.2b).*

(i) ***SPO vs Suboptimality losses:*** *The regret $R(\theta,\theta^\star)$ is indeed the Smart "Predict, then Optimize" loss (SPO) studied in [EG22], with respect to which the circumcenter in Definition 2.1 is shown to be the worst case optimal. However, the symmetric counterpart $R(\theta^\star,\theta)$ coincides with the Suboptimality loss from [Moh+18a], which, as discussed above, is a special case of the ASL connected to the incenter in Definition 2.3.*

(ii) ***Convexity and tractability:*** *The regret R is convex in the second argument [Moh+18a] (note the connection to suboptimality in the previous point), whereas inherently nonconvex in the first argument; see [EG22] for the connection of the latter to the 0-1 loss in classification problems. This observation is indeed aligned with the intractability results of circumcenter (Theorem 2.2) and the tractability of incenter (Corollary 2.6). It is worth noting that [EG22] also proposes a convexified version of the SPO loss (i.e., the regret in the first argument), reminiscent of the hinge loss in classification problems.*

(iii) ***Additional required data measurements****: Given the cost vector θ and the IO data pair $(\hat{s}, x^\star)$ (for the definition of $x^\star$ see (2.2a)), the regret $R(\theta^\star,\theta)$ can be computed without the knowledge of the ground truth $\theta^\star$, whereas its symmetric counterpart $R(\theta,\theta^\star)$ depends explicitly on $\theta^\star$ or its projection through a feature function [EG22].*

2.2.2. GENERAL REFORMULATION FOR MIXED-INTEGER FEASIBLE SETS

In this section, we present a way to reformulate the IO problem using the ASL for problems when $\mathbb{X}(\hat{s})$ is a linear mixed-integer set, which generalizes many formulations from the literature. For this purpose, we consider the mixed-integer feasible set

$$\mathbb{X}(\hat{s}) := \left\{(y,z) \in \mathbb{R}^u \times \mathbb{Z}^v : \hat{A}y + \hat{B}z \leq \hat{c},\ z \in \mathbb{Z}(\hat{w})\right\}, \tag{2.13}$$

where $\hat{s} := (\hat{A}, \hat{B}, \hat{c}, \hat{w})$, and $\mathbb{Z}(\hat{w})$ is a bounded set that may depend on the signal $\hat{w}$. Since y is a continuous variable, $\mathbb{X}(\hat{s})$ has infinitely many elements and we cannot use (2.11) for this IO problem in practice. To solve this problem, one could use first-order iterative methods to directly optimize the IO loss minimization problem (2.10) (we discuss such approaches in Section 2.3). In this section, we leverage classical tools from convex duality to reformulate (2.10) with the ASL and constraint set (2.13) as a tractable finite convex optimization.

For this reformulation, we use the following hypothesis function

$$\langle \theta, \phi(\hat{s}, x) \rangle = \langle y, Q_{yy} y \rangle + \langle y, Q\phi_1(\hat{w}, z) \rangle + \langle q, \phi_2(\hat{w}, z) \rangle, \tag{2.14}$$

where $x := (y, z)$, $\theta := (\mathrm{vec}(Q_{yy}), \mathrm{vec}(Q), q) \in \mathbb{R}^{u^2+um+r}$, $Q_{yy} \succcurlyeq 0$, and $\phi_1 : \mathbb{W} \times \mathbb{Z}^v \to \mathbb{R}^m$ and $\phi_2 : \mathbb{W} \times \mathbb{Z}^v \to \mathbb{R}^r$ are feature functions (ϕ can be written in terms of ϕ_1 and ϕ_2). The choice of a quadratic hypothesis and $\mathbb{X}(\hat{s})$ with linear inequality constraints is chosen because it generalizes the linear hypothesis case, and also for simplicity of exposition. In general, as long as the hypothesis function and constraint set are convex w.r.t. the continuous part of the decision vector (e.g., *conic representable* problems [Moh+18a]), similar results could be derived. Continuing, we use $d(\hat{x}, x) = \|\hat{y} - y\|_\infty + d_z(\hat{z}, z)$, that is, we use the ∞-norm for the continuous part of the decision vector and a general distance function for the integer part of the decision vector. We use the ∞-norm for the continuous part of the decision vector because, to use reformulation techniques based on convex duality, we need the inner maximization problem of the ASL to be concave in y, and by introducing auxiliary integer variables, we can reformulate the ∞-norm as a concave function of y. More precisely, we exploit the identity $\|\hat{y} - y\|_\infty = \max_{h \in \{-1,0,1\}^u, \|h\|_1 = 1} \langle h, \hat{y} - y \rangle$.

Theorem 2.12 (ASL and mixed-integer feasible set)**.** *For the mixed-integer feasible set* (2.13)*, hypothesis function* (2.14)*, and distance function* $d(\hat{x}, x) = \|\hat{y} - y\|_\infty + d_z(\hat{z}, z)$*, problem* (2.10) *can be reformulated as*

$$\begin{aligned}
&\min \kappa \mathscr{R}(\theta) + \frac{1}{N} \sum_{i=1}^{N} \beta_i \\
&\text{s.t. } \theta = (\mathrm{vec}(Q_{yy}), \mathrm{vec}(Q), q) \in \Theta, \quad \lambda_{ijk} \geq 0, \quad \alpha_{ijk}, \beta_i \in \mathbb{R} \\
&\quad \langle \theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]}) \rangle + \alpha_{ijk} + \langle \lambda_{ijk}, \hat{c}_i - \hat{B}_i z_{ij} \rangle - \langle q, \phi_2(\hat{w}_i, z_{ij}) \rangle + \langle h_k, \hat{y}_i \rangle + d_z(\hat{z}_i, z_{ij}) \leq \beta_i \\
&\quad \begin{bmatrix} Q_{yy} & Q\phi_1(\hat{w}_i, z_{ij}) + h_k + \hat{A}_i^\top \lambda_{ijk} \\ * & 4\alpha_{ijk} \end{bmatrix} \succcurlyeq 0,
\end{aligned} \tag{2.15}$$

where the constraints are for all $(i, j, k) \in [N] \times [M_i] \times [2u]$*,* $\mathbb{Z}(\hat{w}_i) := \{z_{i1}, \ldots, z_{ij}, \ldots, z_{iM_i}\}$*,* $M_i := |\mathbb{Z}(\hat{w}_i)|$*, and if* $k \leq u$ *(resp.* $k > u$*),* h_k *is the vector of zeros except for the k'th (resp.* $(k-u)$*'th) element, which is equal to 1 (resp. -1).*

Proof. Section 2.C.4. □

In case we use a linear hypothesis function instead of a quadratic one, the matrix inequality constraints of (2.15) reduce to much simpler linear equality constraints.

Corollary 2.13 (LP reformulation for linear hypotheses)**.** *For the mixed-integer feasible set* (2.13)*, linear hypothesis function (i.e.,* (2.14) *with* $Q_{yy} = 0$*), and distance function*

$d(\hat{x}, x) = \|\hat{y} - y\|_\infty + d_z(\hat{z}, z)$, *problem* (2.10) *can be reformulated as*

$$\begin{aligned}
&\min \kappa\mathscr{R}(\theta) + \frac{1}{N}\sum_{i=1}^{N}\beta_i \\
&\text{s.t. } \theta = (\mathrm{vec}(Q), q) \in \Theta, \quad \lambda_{ijk} \geq 0, \quad \beta_i \in \mathbb{R} \\
&\quad \langle\theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]})\rangle + \langle\lambda_{ijk}, \hat{c}_i - \hat{B}_i z_{ij}\rangle - \langle q, \phi_2(\hat{w}_i, z_{ij})\rangle + \langle h_k, \hat{y}_i\rangle + d_z(\hat{z}_i, z_{ij}) \leq \beta_i \\
&\quad Q\phi_1(\hat{w}_i, z_{ij}) + h_k + \hat{A}_i^\top \lambda_{ijk} = 0,
\end{aligned} \tag{2.16}$$

where the constraints are for all $(i, j, k) \in [N] \times [M_i] \times [2u]$, $\mathbb{Z}(\hat{w}_i) := \{z_{i1}, \ldots, z_{ij}, \ldots, z_{iM_i}\}$, $M_i := |\mathbb{Z}(\hat{w}_i)|$, *and if* $k \leq u$ *(resp.* $k > u$*),* h_k *is the vector of zeros except for the k'th (resp.* $(k-u)$*'th) element, which is equal to 1 (resp. -1).*

We note that one can reduce the number of constraints in these reformulations by a factor of $2u$ by using $d(\hat{x}, x) = d_z(\hat{z}, z)$, i.e., only penalizing the integer part of the decision vector, which is equivalent to setting $h_k = 0\ \forall k \in [2u]$.

Remark 2.14 (Generality of Theorem 2.12)**.** *Reformulation* (2.15) *in Theorem 2.12, for IO problem with mixed-integer decision sets, generalizes several reformulations from the literature. For instance, for an IO problem with purely continuous decision sets and linear hypothesis functions, reformulation* (2.15) *with* $d_z(\hat{x}, x) = 0$ *and* $h_k = 0\ \forall k \in [2u]$ *reduces to classical Inverse Linear Optimization reformulations [AO01; CLT19]. Similarly, if the IO problem has purely continuous decision set and a quadratic hypothesis function, then the program* (2.15) *with* $d_z(\hat{x}, x) = 0$ *and* $h_k = 0\ \forall k \in [2u]$ *reduces to the LMI reformulation in [AKE21]. If the decision set is purely discrete, that is,* $\mathbb{X}(\hat{s})$ *has finitely many elements, then* (2.15) *and* (2.16) *recover* (2.11).

In conclusion, by dualizing the continuous part of the problem, Theorem 2.12 and Corollary 2.13 present a way to deal with a case when the feasible set $\mathbb{X}(\hat{s})$ is a mixed-integer set. It is worth noting that when the feasible set $\mathbb{X}(\hat{s})$ is a mixed-integer set, the size of reformulations (2.15) and (2.16) grow linearly in the number of integer variables, in the dimension of the continuous variables, and in the size of the dataset. That means, for IO problems with large datasets and/or mixed-integer sets with a large number of decision variables, these reformulations can be computationally challenging to solve. This issue is the main focus of our tailored first-order algorithm in the next section.

2.3. TAILORED ALGORITHM: STOCHASTIC APPROXIMATE MIRROR DESCENT

All approaches to solving IO problems presented so far depend on optimization programs with possibly a large number of constraints. Given a dataset $\{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$, these optimization problems have at least $\sum_{i=1}^N |\mathbb{X}(\hat{s}^{[i]})|$ (or $\sum_{i=1}^N |\mathbb{Z}(\hat{w}_i)|$ for the mixed-integer case) constraints. When N is too large (i.e., we have too many signal-response examples) or $|\mathbb{X}(\hat{s}^{[i]})|$ is too large, solving these optimization programs may be intractable in practice. Thus, the focus of this section is to develop a tailored first-order algorithm to tackle such IO problems in a provably efficient way, opening doors to a wider range

of real-world applications. A straightforward first-order approach to solve (2.10) is the standard *Project Subgradient method* [Sho85]. This is a general first-order optimization algorithm, and it converges to an optimal solution of the problem under mild convexity conditions on the objective function and constraint set. For example, this was the approach used in [RBZ06] to solve the Maximum Margin Planning problem (see Remark 2.10). Another idea is to solve (2.10) as a minimax problem. [Tas+06] showed that in some special cases, this problem can be formulated as a bilinear saddle point problem, and then used Nesterov's dual extragradient method [Nes07] to solve it. However, other than only being applicable to a restrictive class of problems, this approach also requires $2N$ projections onto $\mathbb{X}(\hat{s}^{[i]})$ to be computed per iteration of the algorithm, which may be prohibitive in many applications. Other algorithms for minimax problems, such as Nemirovski's Mirror-Prox algorithm [Nem04], would suffer from similar drawbacks. In this section, we introduce an efficient optimization algorithm that can be applied to general IO problems, while also exploiting the specific structures that stem from IO problems. To this end, we define the following notion of a stochastic approximate subgradient.

Definition 2.15 (Stochastic approximate subgradient)**.** *Let $f : \Theta \to \mathbb{R}$ be a convex function. We say that the random vector $\tilde{g}_\varepsilon(\theta)$ is a* stochastic approximate subgradient *of f at θ if $\mathbb{E}\left[\tilde{g}_\varepsilon(\theta) \mid \theta\right] = g_\varepsilon(\theta)$ and $g_\varepsilon(\theta)$ is an ε-subgradient of f for some $\varepsilon \geq 0$, that is,*

$$f(\theta) - f(\nu) \leq \langle g_\varepsilon(\theta), \theta - \nu \rangle + \varepsilon, \quad \forall \nu \in \Theta.$$

For different notions of approximate subgradients, see [RBZ07; TSK22]. Next, we propose to solve IO problems with a novel algorithm, the *Stochastic Approximate Mirror Descent* (SAMD). We first define the SAMD as an algorithm to optimize general convex programs and prove convergence rates for it. After that, we discuss how we can use the SAMD algorithm to exploit the structure of IO problems. For a function f and set Θ, the SAMD updates are

$$\theta_{t+1} = \operatorname*{argmin}_{\theta \in \Theta} \left\{ \eta_t \langle \tilde{g}_{\varepsilon_t}(\theta_t), \theta \rangle + \mathscr{B}_\omega(\theta, \theta_t) \right\}, \tag{SAMD}$$

where η_t is the step-size, the function $\mathscr{B}_\omega$ is the Bregman divergence w.r.t. $\omega : \Theta \to \mathbb{R}$ [Bub15, Section 4], and $\tilde{g}_{\varepsilon_t}(\theta_t)$ is a *stochastic approximate subgradient* of f as defined in Definition 2.15. The SAMD algorithm can be interpreted as a combination of a stochastic mirror descent algorithm [Bub15, Section 6.1] and an ε-subgradient method [Ber15, Section 3.3]. Next, we prove a convergence rate for the SAMD algorithm, where we use the concept of *relative strong convexity*, which is a generalization of the standard strong convexity property [LFN18]. Namely, if $\mathscr{R}$ is α-strongly convex relative to $\mathscr{B}_\omega$, then $\mathscr{R}(x) - \mathscr{R}(y) \leq \langle g(x), x - y \rangle - \alpha \mathscr{B}_\omega(y, x)$, where $g(x) \in \partial \mathscr{R}(x)$.

Proposition 2.16 (SAMD convergence rate)**.** *Let $f : \Theta \to \mathbb{R}$ be a convex function, and Θ be a convex set. Assume $\mathscr{B}_\omega(\theta, \nu) \leq R^2$, for some $R > 0$, and $\mathbb{E}\left[\|\tilde{g}_{\varepsilon_t}(\theta)\|_*^2 \mid \theta\right] \leq G^2$, $\forall \theta, \nu \in \Theta$. Using $\eta_t = c/\sqrt{t}$ for some constant $c > 0$, the SAMD algorithm guarantees*

$$\mathbb{E}\left[f\left(\frac{1}{T} \sum_{t=1}^{T} \theta_t \right) \right] - \min_{\theta \in \Theta} f(\theta) \leq \left(\frac{R^2}{c} + cG^2 \right) \frac{1}{\sqrt{T}} + \frac{1}{T} \sum_{t=1}^{T} \varepsilon_t.$$

Moreover, if we assume $f = h + \mathscr{R}$, where h is convex and $\mathscr{R}$ is α-strongly convex relative to $\mathscr{B}_\omega$, then using $\eta_t = 2/\alpha(t+1)$, the SAMD algorithm guarantees

$$\mathbb{E}\left[f\left(\frac{2}{T(T+1)}\sum_{t=1}^{T} t\theta_t\right)\right] - \min_{\theta\in\Theta} f(\theta) \le \frac{2G^2}{\alpha(T+1)} + \frac{2}{T(T+1)}\sum_{t=1}^{T} t\varepsilon_t.$$

Proof. Section 2.C.5. □

The convergence rate of the SAMD algorithm is a combination of the $O(1/\sqrt{T})$ (or $O(1/T)$) rate of the stochastic mirror descent algorithm plus a term that depends on $\{\varepsilon_t\}_{t=1}^T$ due to the use of approximate subgradients. Notice that, if we use the SAMD algorithm with errors ε_t diminishing at a rate $O(1/t)$ (i.e., as t increases, we use increasingly more precise approximate subgradients), then the convergence rates of Proposition 2.16 reduce to $O(1/\sqrt{T})$ and $O(1/T)$. The SAMD algorithm differs from a simple projected subgradient method in three major ways: it uses (i) mirror descent updates, (ii) stochastic subgradients, and (iii) approximate subgradients. Next, we discuss how each of these properties can be used to exploit the structure of the IO problem (2.10).

2.3.1. MIRROR DESCENT UPDATES

For instance, consider (2.10) with $\mathscr{R}(\theta) = \|\theta\|_1$ and $\Theta = \mathbb{R}^p$. This problem can be equivalently written as

$$\begin{aligned} \min_{\theta\in\mathbb{R}^p} \quad & \frac{1}{N}\sum_{i=1}^{N} \ell_\theta(\hat{s}^{[i]}, \hat{x}^{[i]}) \\ \text{s.t.} \quad & \tilde{\kappa}\|\theta\|_1 \le 1, \end{aligned} \tag{2.17}$$

for some $\tilde{\kappa}$ that depends on κ and the data $\{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$. Then, by introducing the nonnegative variables θ^+ and θ^-, defining $\tilde{\theta} := (\theta^+, \theta^-)$ (i.e., the concatenation of θ^+ and $\theta-$), and setting $\theta = \theta^+ - \theta^- = [I\ -I]\tilde{\theta}$ (here, I is the identity matrix and $[I\ -I]$ is a block matrix), (2.17) can be written as

$$\min_{\tilde{\theta}\in\Delta_{\tilde{\kappa}}} \frac{1}{N}\sum_{i=1}^{N} \ell_{[I\ -I]\tilde{\theta}}(\hat{s}^{[i]}, \hat{x}^{[i]}), \tag{2.18}$$

where $\Delta_{\tilde{\kappa}} := \{\tilde{\theta} \in \mathbb{R}^{2p} : \tilde{\kappa}\|\tilde{\theta}\|_1 \le 1, \tilde{\theta} \ge 0\}$ [Tib96]. In other words, (2.17) can be recast as an optimization problem over a simplex. Next, choosing $\omega(\theta) = \sum_{i=1}^p \theta_i \log(\theta_i)$, the SAMD updates applied to (2.18) can be written as "exponentiated updates"

$$\tilde{\theta}_{t+1} = \begin{cases} \tilde{\theta}_t \odot \exp(-\eta_t \tilde{g}_{\varepsilon_t}(\tilde{\theta}_t)) & \text{if} \quad \tilde{\kappa}\|\tilde{\theta}_t \odot \exp(-\eta_t \tilde{g}_{\varepsilon_t}(\tilde{\theta}_t))\|_1 \le 1 \\ \frac{\tilde{\theta}_t \odot \exp(-\eta_t \tilde{g}_{\varepsilon_t}(\tilde{\theta}_t))}{\tilde{\kappa}\|\tilde{\theta}_t \odot \exp(-\eta_t \tilde{g}_{\varepsilon_t}(\tilde{\theta}_t))\|_1} & \text{otherwise.} \end{cases}$$

Exponentiated updates are known to have better convergence properties compared to standard subgradient descent updates for "simplex constrained" problems, as well as not requiring solving optimization problems to project onto the simplex [Bub15]. For a detailed discussion on the advantages of mirror descent updates for different settings,

see [JN11, Section 5.7]. In general, by choosing $\omega(\theta) = \frac{1}{2}\|\theta\|_2^2$, the SAMD updates reduce to standard project subgradient updates

$$\theta_{t+1} = \Pi_\Theta(\theta_t - \eta_t \tilde{g}_{\varepsilon_t}(\theta_t)),$$

that is, the subgradient method can be seen as a special case of the mirror descent algorithm.

2.3.2. STOCHASTIC SUBGRADIENTS

Using stochastic subgradients is advantageous when computing a stochastic subgradient is computationally cheaper than computing a deterministic subgradient. This observation is supported by the fact that, in expectation, the stochastic subgradient method guarantees the same convergence rate as its deterministic version [Bub15, Theorem 6.1]. For the IO loss minimization problem (2.10), computing a stochastic subgradient can be significantly cheaper than computing the full subgradient, due to the finite sum structure of (2.10). Namely, by Danskin's theorem [Ber08, Section B.5], we have that the subdifferential of the ASL w.r.t. θ is

$$\partial \ell_\theta(\hat{s}, \hat{x}) = \operatorname{conv}\left\{ \phi(\hat{s}, \hat{x}) - \phi(\hat{s}, x^\star(\hat{s})) \;\middle|\; x^\star(\hat{s}) \in \operatorname*{argmax}_{x \in \mathbb{X}(\hat{s})} \left\{ d(\hat{x}, x) - \langle \theta, \phi(\hat{s}, x) \rangle \right\} \right\}. \tag{2.19}$$

Thus, to compute a subgradient of $\frac{1}{N}\sum_{i=1}^N \ell_\theta(\hat{s}^{[i]}, \hat{x}^{[i]})$, we need to solve N maximization problems, one for each signal-response pair $\{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$. On the other hand, by sampling an index j uniformly from $[N]$, we have that $g_j(\theta) \in \partial \ell_\theta(\hat{s}_j, \hat{x}_j)$ is a stochastic subgradient of $\frac{1}{N}\sum_{i=1}^N \ell_\theta(\hat{s}^{[i]}, \hat{x}^{[i]})$, that is, $\mathbb{E}_{j\sim[N]}[g_j(\theta) \mid \theta] = (1/N)\sum_{i=1}^N g_i(\theta)$. This is a standard method for computing stochastic (sub)gradients for empirical risk minimization-type problems (e.g., [Bub15, Chapter 6]). In summary, to compute an unbiased stochastic subgradient of (2.10), instead of N, we need to solve only *one* maximization problem.

2.3.3. APPROXIMATE SUBGRADIENTS

As shown in (2.19), we need to solve the optimization problem $\max_{x\in\mathbb{X}(\hat{s})}\{d(\hat{x}, x) - \langle \theta, \phi(\hat{s}, x)\rangle\}$ in order to compute a subgradient of the ASL. However, in practice, it may be too costly to solve this optimization problem to optimality at each iteration of the algorithm. Thus, it would be useful if we could use an approximate solution to this problem, instead of an optimal one. Turns out that, indeed, given an *ε-approximate solution* to the maximization problem, that is, a feasible point x_ε such that

$$d(\hat{x}, x_\varepsilon) - \langle \theta, \phi(\hat{s}, x_\varepsilon)\rangle \geq \max_{x\in\mathbb{X}(\hat{s})} \left\{ d(\hat{x}, x) - \langle \theta, \phi(\hat{s}, x)\rangle \right\} - \varepsilon,$$

we can construct an ε-subgradient of the ASL.

Lemma 2.17 (Approximate solutions and ε-subgradients)**.** *Let x_ε be a feasible, ε-suboptimal solution of $\max_{x\in\mathbb{X}(\hat{s})} \left\{ d(\hat{x}, x) - \langle \theta, \phi(\hat{s}, x)\rangle \right\}$. Thus, the vector $g_\varepsilon(\theta) = \phi(\hat{s}, \hat{x}) - \phi(\hat{s}, x_\varepsilon)$ is an ε-subgradient of $\ell_\theta(\hat{s}, \hat{x})$ with respect to θ, i.e.,*

$$\ell_\theta(\hat{s}, \hat{x}) - \ell_\nu(\hat{s}, \hat{x}) \leq \langle \phi(\hat{s}, \hat{x}) - \phi(\hat{s}, x_\varepsilon), \theta - \nu\rangle + \varepsilon.$$

Proof. Section 2.C.6. □

In summary, we can use an approximate solution to the inner maximization problem of the ASL to construct ε-subgradients, and as proved in Proposition 2.16, these can be effectively used to compute an approximate solution of convex optimization problems. Finally, it is not difficult to show that by combining stochastic subgradients (i.e., by using only one random signal-response pair) and approximate subgradients (i.e., solving the respective maximization problem approximately), one gets a stochastic approximate subgradient (Definition 2.15). Putting these ideas together, Algorithm 1 shows the pseudocode of the SAMD algorithm applied to problem (2.10), where we denote the gradient (or a subgradient) of $\mathscr{R}$ as $\nabla\mathscr{R}$. In line 3 of Algorithm 1, one example is sampled from the dataset for each iteration of the algorithm. However, in practice, it may be advantageous to instead sample a batch of $1 \leq B \leq N$ examples, and use this batch of data to compute the approximate stochastic subgradient. By changing the size B of the batch, we can control the trade-off between having more precise subgradients (large B) versus faster subgradient computations (small B). This idea is explored in the numerical experiments of Section 2.4.4.

Algorithm 1 SAMD algorithm for (2.10)

1: **Input:** Step-size sequence $\{\eta_t\}_{t=1}^T$, κ, θ_1, d, ϕ, ω, $\nabla\mathscr{R}$ and dataset $\{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$.
2: **for** $t = 1, \ldots, T$ **do**
3: Sample j uniformly from $\{1, \ldots, N\}$
4: Compute x_t, a (possibly approximate) solution of $\max_{x \in \mathbb{X}(\hat{s}^{[j]})} \{d(\hat{x}^{[j]}, x) - \langle \theta_t, \phi(\hat{s}^{[j]}, x)\rangle\}$
5: Approximate stochastic subgradient: $\tilde{g}_t(\theta_t) = \kappa\nabla\mathscr{R}(\theta_t) + \phi(\hat{s}^{[j]}, \hat{x}^{[j]}) - \phi(\hat{s}^{[j]}, x_t)$
6: Mirror descent step: $\theta_{t+1} = \operatorname{argmin}_{\theta\in\Theta} \{\eta_t \langle \tilde{g}_t(\theta_t), \theta\rangle + \mathscr{B}_\omega(\theta, \theta_t)\}$
7: **end for**
8: **Output:** $\{\theta_t\}_{t=1}^T$

We end this section by briefly discussing the case of *online IO*. In this scenario, instead of having a dataset of signal-response data upfront, we receive one signal-response pair at a time. After receiving each signal-response data pair, we must choose a cost vector θ_t using all the information gathered so far, and evaluate the loss of this cost vector. The final performance of the algorithm is measured using *regret* performance metrics.

Remark 2.18 (Regret bounds and online IO)**.** *[BPS17] use an online Multiplicative Weights Updates (MWU) algorithm to prove an $O(\sqrt{T})$ regret bound, and more recently, [BFL23] use an online adaptation of the circumcenter concept to prove an $O(\log(T))$ regret bound. In Algorithm 1, line 3, if instead of sampling a new signal-response example we use the online data received at that iteration, Algorithm 1 can be readily adapted to online IO problems. Moreover, the convergence bound of Proposition 2.16 may also be straightforwardly converted to an $O(\sqrt{T})$ regret bound (or $O(\log(T))$ for the strongly convex case) [Ora19, Chapter 3]. In particular, Algorithm 1 can be interpreted as a generalization of the MWU algorithm [AO14, Appendix A.2].*

2.4. NUMERICAL EXPERIMENTS

In this section, we numerically evaluate the approaches to IO proposed in this chapter. All linear and quadratic programs were solved using Gurobi. All semidefinite programs were solved using CVXPY with MOSEK as the solver. For all results presented in this section, we learn the expert's cost function using a training dataset of expert signal-response data and evaluate its performance using a test dataset (i.e., out-of-sample performance). Moreover, in every plot, we report the average performance value for 10 randomly generated true cost vectors, as well as the 5th and 95th percentile bounds. In Appendix 2.D.2, the in-sample results are reported, complementing the ones from this section. All of our experiments are reproducible and are part of the InvOpt Python package [Zat23b].

2.4.1. CONSISTENT DATA

In this section, we numerically evaluate the approaches discussed in Section 2.1 for IO problems with consistent data.

Decision problem of the expert. To generate its decisions, the expert solves the binary linear program

$$\begin{aligned} \min_{x} \quad & \langle \theta, x \rangle \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \{0,1\}^n, \end{aligned} \tag{2.20}$$

where $\theta \in \mathbb{R}^n_+$ is the cost vector, $A \in \mathbb{R}^{t \times n}$, $b \in \mathbb{R}^t$, and x is the decision vector. Although the decision problem (2.20) is presented as a general binary linear program, many real-world problems can be modeled within this class of optimization problems. As a motivating example, we briefly discuss the problem of modeling consumer behavior, and how it fits (2.20). In this problem, the expert is a consumer, who given some contextual information, decides which products to buy from a set of n products. That is, each component of the decision vector $x \in \{0,1\}^n$ corresponds to one of the n products, and equals 1 if the consumer buys it, and 0 otherwise. The signal with contextual information could be, for instance, $\hat{s} = (A, b)$, where each component of $A \in \mathbb{R}^n$ corresponds to the price of each product, and $b \in \mathbb{R}$ corresponds to the total budget of the consumer. In this case, the constraint $Ax \leq b$ simply represents the budget constraint of the consumer. Finally, each component of the cost vector $\theta \in \mathbb{R}^n$ represents (the negative of) the utility the consumer assigns to each product, and by minimizing $\langle \theta, x \rangle$, the goal of the consumer is to buy the set of products that maximizes its utility, while respecting its budget constraint. Thus, in this context, solving the IO problem translates to learning the utility function underlying the actions of a consumer agent.

Data generation. To generate training and test data, we sample cost vectors uniformly from $\{\theta \in \mathbb{R}^n : 0 \leq \theta \leq \mathbb{1}\}$, we sample $\hat{A}$ uniformly from $\{A \in \mathbb{R}^{t \times n} : -1 \leq A \leq 0\}$ and $\hat{b}$ uniformly from $\{b \in \mathbb{R}^t : -\mathbb{1} \leq b \leq 0\}$. To make sure the problem instances are feasible, we check if the sum of each row of $\hat{A}$ is larger than the respective component of $\hat{b}$, which ensures that $x = (1, \ldots, 1)$ is a feasible solution of (2.20). Given a signal $\hat{s} = (\hat{A}, \hat{b})$, we generate a response $\hat{x}$ by solving (2.20), i.e., $\hat{x} \in \operatorname{argmin}_{x \in \mathbb{X}(\hat{s})} \langle \theta, x \rangle$. To evaluate the IO approaches, we generate 10 random cost vectors θ_{true}. For each of these cost vectors, we generate a training dataset $\widehat{\mathscr{D}}_{\text{train}} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$ and a test dataset $\widehat{\mathscr{D}}_{\text{test}} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$, with $N = 100$, $n = 6$ and $t = 4$.

Approach	Feasibility	Incenter	Circumcenter	Ellip. incenter
Time (seconds)	69	98	1323	915
Approach	Ellip. circ.	Cutting plane	Predic. loss	
Time (seconds)	1345	206	236	

Table 2.1: Computational time to generate the results of Figure 2.3.

IO approaches. We compare seven approaches to choose a vector from the set of consistent cost vectors (1.4):

- **Feasibility**: we use (2.1) with $\|\theta\|_1 = 1$, $s = (A, b)$, $\mathbb{X}(s) = \{x \in \{0,1\}^n : Ax \leq b\}$, $\phi(x, s) = x$, augmented with the constraint $\theta \in \Theta = \{\theta \in \mathbb{R}^n : \theta \geq 0\}$;
- **Incenter**: we use (2.9) with $\mathscr{R}(\theta) = \frac{1}{2}\|\theta\|_2^2$, $s = (A, b)$, $\mathbb{X}(s) = \{x \in \{0,1\}^n : Ax \leq b\}$, $\phi(x, s) = x$, $d(\hat{x}^{[i]}, x^{[i]}) = \|\hat{x}^{[i]} - x^{[i]}\|_2$, and $\Theta = \{\theta \in \mathbb{R}^n : \theta \geq 0\}$;
- **Circumcenter**: we solve the following optimization problem:

$$\begin{aligned} \min_{\|\theta\|_2=1} \quad & r \\ \text{s.t.} \quad & \max_{\substack{\tilde{\theta}\in\mathbb{C} \\ \|\tilde{\theta}\|_2=1}} \|\theta - \tilde{\theta}\|_2^2 \leq r, \end{aligned} \tag{2.21}$$

  which is an epigraph reformulation of the circumcenter problem (see the proof of Theorem 2.2). To solve this optimization problem, we substitute the max constraint by the constraints $\|\theta_E - \tilde{\theta}\|_2^2 \leq r,\ \forall \theta_E \in E$, where E is the set of normalized extreme points of $\mathbb{C}$;
- **Ellip. incenter**: we solve (2.8), the ellipsoidal version of the incenter concept. In our implementation, we use the convex constraint $\|\theta\| \leq 1$ instead of $\|\theta\| = 1$, since one can show that the constraint $\|\theta\| \leq 1$ will always be tight at an optimum when $\text{int}(\mathbb{C}) \neq \emptyset$;
- **Ellip. circumcenter**: we solve

$$\begin{aligned} \max_{A\succcurlyeq 0,\theta} \quad & \log\det(A) \\ \text{s.t.} \quad & \max_{\substack{\tilde{\theta}\in\mathbb{C} \\ \|\tilde{\theta}\|_2=1}} \|A(\tilde{\theta} - \theta)\|_2^2 \leq 1, \end{aligned}$$

  which is an ellipsoidal generalization of the circumcenter reformulation (2.21);
- **Cutting plane**: we use the cutting plane algorithm of [Wan09];
- **Predictability loss**: we use the predictability loss of [ASS18] (Appendix 2.A).

Results. Figure 2.3 shows the results for this scenario. To make the comparisons consistent, before evaluating the results, all cost vectors are normalized. For all the plots, the x-axis refers to the number of training examples used to compute the results. The idea

is to evaluate how efficient these approaches are with respect to the amount of data fed to them. Figure 2.3a shows the difference between the cost vector returned by the IO approaches (which we name θ_{IO}) and the cost vector used to generate the data (θ_{true}). As expected, the more data we feed to the approaches, the closer θ_{IO} tends to get from θ_{true}. Comparing them, the incenter, ellipsoidal incenter, and ellipsoidal circumcenter approaches have similar performance and clearly outperform the other approaches. Figure 2.3b shows the average difference between the optimal decision of (2.20) using θ_{IO} (which we name x_{IO}) and the decision in the test dataset (which we name x_{true}). Again, we see that the incenter, ellipsoidal incenter, and ellipsoidal circumcenter approaches present the best performance. Figure 2.3c shows the normalized difference between the cost of the expert decisions and the cost of the decisions using θ_{IO}. More precisely, we define $\text{Cost}_{\text{IO}} := \sum_{i=1}^{N} \langle \theta_{\text{true}}, x_{\text{IO},i} \rangle$ and $\text{Cost}_{\text{true}} := \sum_{i=1}^{N} \langle \theta_{\text{true}}, \hat{x}^{[i]} \rangle$ and compare the relative difference between them. Notice that this difference will always be nonnegative by the optimality of $\hat{x}^{[i]}$. Once again, the incenter, ellipsoidal incenter, and ellipsoidal circumcenter approaches outperform the other approaches.

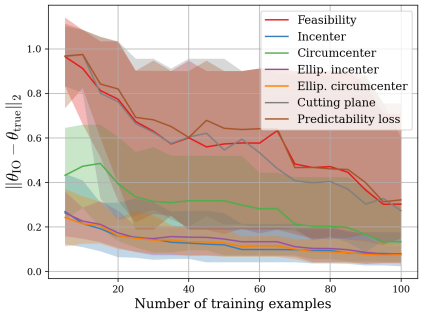
Moreover, Table 2.1 shows the time it took to generate the results of this section for each approach. As can be seen, even though the incenter, ellipsoidal incenter, and ellipsoidal circumcenter approaches show similar performance, the incenter approach is at least one order of magnitude faster to compute. Although these numbers depend on the actual implementation of each IO approach, such a difference in solving time is expected, since the incenter problem can be formulated as a quadratic program, whereas the ellipsoidal approaches involve semidefinite constraints. Finally, regarding the main driver for improved out-of-sample performance of the Incenter and the ellipsoidal approaches, one possible explanation is that these approaches optimize for a vector in the interior of $\mathbb{C}$, that is, away from the boundaries of this set. This conclusion is based on the following observations: recall the intuition for the incenter vector provided in Remark 2.4, also visualized in Figure 2.1b. That is, the idea behind the incenter is to find the vector furthest away from the boundaries of the set $\mathbb{C}$. Also, recall that [BFL23] showed that circumcenter concept fails for online IO problems precisely because, in the worst case, the circumcenter vector can lie exactly in the boundary of the set $\mathbb{C}$ [BFL23, Figure 2], and the ellipsoidal circumcenter addresses this problem since its circumcenter lies inside the set $\mathbb{C}$ [BFL23, Figure 3]. Therefore, the incenter, ellipsoidal incenter, and ellipsoidal circumcenter in some sense optimize for vector in the interior of $\mathbb{C}$, and as shown in Figure 2.3, achieve similar out-of-sample performance in this offline IO experiment (although the incenter vector is computationally cheaper to compute).

2.4.2. INCONSISTENT DATA

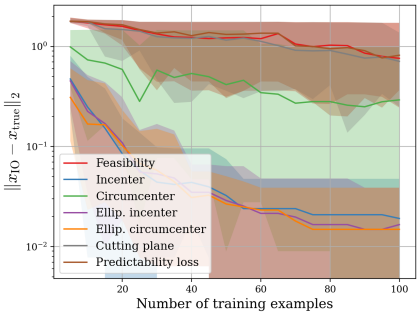
In this section, we numerically evaluate the approaches discussed in Section 2.2 for IO problems with inconsistent data.

Decision problem of the expert. To generate its decisions, the expert solves the binary linear program (2.20), where $\theta \in \mathbb{R}^n$ is the cost vector, $A \in \mathbb{R}^{t \times n}$ and $b \in \mathbb{R}^t$, and x is the decision vector. Notice that different from the previous section, we do not assume the cost vector θ is nonnegative.

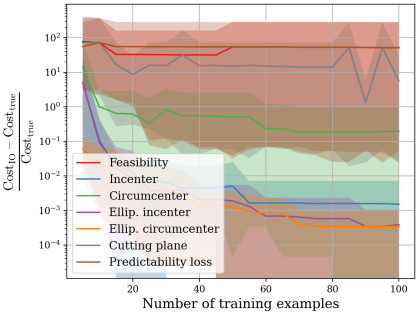
Data generation. To generate training and test data, we sample cost vectors uniformly from $\{\theta \in \mathbb{R}^n : -\mathbb{1} \leq \theta \leq \mathbb{1}\}$, we sample $\hat{A}$ uniformly from $\{A \in \mathbb{R}^{t \times n} : -1 \leq A \leq 1\}$



(a) Difference between the true cost vector (θ_{true}) and the one learned using IO (θ_{IO}).



(b) Average error between the decision generated by θ_{true} and θ_{IO}.



(c) Relative difference between the cost of the decisions generated using θ_{true} and θ_{IO}.

Figure 2.3: Out-of-sample results for consistent data scenario.

and $\hat{b}$ uniformly from $\{b \in \mathbb{R}^t : -\mathbb{1} \leq b \leq 0\}$. After generating a signal $\hat{s} = (\hat{A}, \hat{b})$, we check if $\mathbb{X}(\hat{s})$ is nonempty to ensure the problem instance is feasible. To generate the response vectors $\hat{x}$ for the training datasets, we solve problem (2.20) with noise added to the cost vector, i.e., $\hat{x} \in \operatorname{argmin}_{x \in \mathbb{X}(\hat{s})} \langle \theta + w, x \rangle$, where $w \in \mathbb{R}^n$ is a random vector with components sampled from a normal distribution with zero mean and standard deviation equal to 0.05. This means that different from the consistent data case, (most probably) there will be no single cost vector consistent with the entire training dataset. To evaluate the IO approaches, we generate 10 random cost vectors θ_{true}. For each of these cost vectors, we generate a training dataset $\widehat{\mathscr{D}}_{\text{train}} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$ (by solving the noisy version of (2.20), with a different noise vector w for each signal-response pair) and a test dataset $\widehat{\mathscr{D}}_{\text{test}} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$ (noiseless), with $N = 100$, $n = 10$ and $t = 8$.

IO approaches. We compare five IO approaches for this problem:

- **Suboptimality Loss (SL)**: we use (2.10) with $\kappa = 0$, $\Theta = \mathbb{R}^n$, $s = (A, b)$, $\mathbb{X}(s) = \{x \in \{0,1\}^n : Ax \leq b\}$, $\phi(x, s) = x$, and the suboptimality loss $\ell_\theta(\hat{s}, \hat{x}) = \max_{x \in \mathbb{X}(\hat{s})} \{\langle \theta, \phi(\hat{s}, \hat{x}) - \phi(\hat{s}, x) \rangle\}$. To prevent the trivial solution $\theta = 0$, we add a norm equality constraint $\|\theta\|_\infty = 1$, and solve $2n$ linear programs, one for each facet of the ∞-norm unit sphere [Moh+18a];

Approach	SL	ASL	Ellipsoidal ASL	Cutting plane	Predictability loss
Time (seconds)	230	241	351	440	6192

Table 2.2: Computational time to generate the results of Figure 2.4.

- **Augmented Suboptimality Loss (ASL)**: we use (2.10) with the ASL, $\kappa = 0.001$, $\mathscr{R}(\theta) = \frac{1}{2}\|\theta\|_2^2$, $s = (A, b)$, $\mathbb{X}(s) = \{x \in \{0,1\}^n : Ax \leq b\}$, $\phi(x, s) = x$, $d(\hat{x}^{[i]}, x^{[i]}) = \|\hat{x}^{[i]} - x^{[i]}\|_2$, and $\Theta = \mathbb{R}^n$;
- **Ellipsoidal ASL**: we solve

$$\begin{aligned} \min_{A,\theta,\beta_1,\ldots,\beta_N} \; & -\kappa \log(\det(A)) + \frac{1}{N}\sum_{i=1}^{N} \beta_i \\ \text{s.t.} \quad & \langle \theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]}) - \phi(\hat{s}^{[i]}, x^{[i]})\rangle + \|A(\phi(\hat{s}^{[i]}, x^{[i]}) - \phi(\hat{s}^{[i]}, \hat{x}^{[i]}))\|_2 \leq \beta_i, \\ & \qquad\qquad \forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]}), \forall i \in [N] \\ & \|\theta\|_2 \leq 1, \quad A \succcurlyeq 0, \end{aligned} \tag{2.22}$$

  which is an ellipsoidal generalization of (2.11) (see Remark 2.7).
- **Cutting plane**: we use the cutting plane method of [BCZ22], which is a extension of the cutting-plane algorithm of [Wan09] for IO problems with inconsistent data;
- **Predictability loss**: we use the predictability loss of [ASS18] (Appendix 2.A).

Notice that we do not test circumcenter-based approaches since they are not defined for the inconsistent data case, that is, when $\mathbb{C} \setminus \{0\} = \emptyset$.

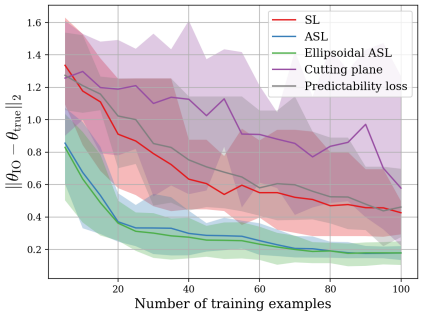
Results. Figure 2.4 shows the results this scenario. The discussion on the interpretation of each performance metric presented in Figure 2.4 mirrors the one of Figure 2.3. Importantly, the approach based on the ASL and its ellipsoidal version outperforms the other approaches. Moreover, as expected, due to the noise in the training data, one can see that θ_{IO} was not able to reproduce the behavior of the expert as well as in the consistent (i.e., noiseless) case. In Table 2.1 we show the time it took to generate the results of this section for each approach. As can be seen, even though the ASL and Ellipsoidal ASL show similar performance, the ASL approach is more computationally efficient. Again, this difference is because the Ellipsoidal ASL involves semidefinite constraints (Eq. (2.22)), which is not the case for the standard ASL (Eq. (2.11)).

2.4.3. MIXED-INTEGER FEASIBLE SET

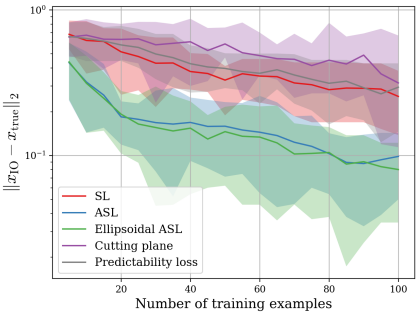
In this section, we numerically evaluate the approach from Section 2.2.2 for IO problems with mixed-integer feasible sets.

Decision problem of the expert. To generate its decisions, the expert solves a mixed-integer linear program of the form

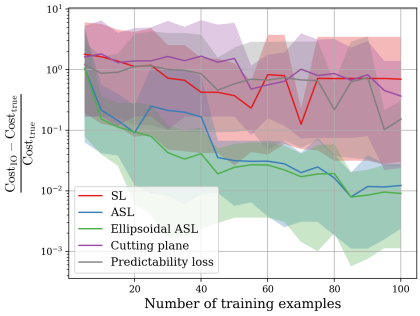
$$\begin{aligned} \min_{y,z} \quad & \langle q_y, y\rangle + \langle q_z, z\rangle \\ \text{s.t.} \quad & Ay + Bz \leq c \\ & 0 \leq y \leq \mathbb{1}, \; z \in \{0,1\}^v, \end{aligned} \tag{2.23}$$



(a) Difference between the true cost vector (θ_{true}) and the one learned using IO (θ_{IO}).



(b) Average error between the decision generated by θ_{true} and θ_{IO}.



(c) Relative difference between the cost of the decisions generated using θ_{true} and θ_{IO}.

Figure 2.4: Out-of-sample results for inconsistent data scenario.

where $\theta := (q_y, q_z) \in \mathbb{R}^{u+v}_+$ is the cost vector $A \in \mathbb{R}^{t\times u}$, $B \in \mathbb{R}^{t\times v}$ and $b \in \mathbb{R}^t$.

Data generation. To generate training and test data, we sample the cost vector uniformly from $\{\theta \in \mathbb{R}^{u+v} : 0 \le \theta \le \mathbb{1}\}$, we sample $\hat{A}$ uniformly from $\{A \in \mathbb{R}^{t\times u} : -1 \le A \le 0\}$, $\hat{B}$ uniformly from $\{B \in \mathbb{R}^{t\times v} : -1 \le B \le 0\}$ and $\hat{c}$ uniformly from $\{c \in \mathbb{R}^t : -2 \le c \le 0\}$. To make sure the problem instances are feasible, we checked if the sum of each row of $[\hat{A}\ \hat{B}]$ is smaller than the respective component of $\hat{c}$. Given a tuple $(\hat{A}, \hat{B}, \hat{c})$, we generate a response $\hat{x} = (\hat{y}, \hat{z})$ by solving problem (2.23). To evaluate the IO approaches, we generate 10 random cost vectors θ_{true}. For each of these cost vectors, we generate a training dataset $\widehat{\mathcal{D}}_{train} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$ and a test dataset $\widehat{\mathcal{D}}_{test} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$, with $N = 100$, $u = v = 6$ and $t = 4$.

IO approaches. We compare six IO approaches for this problem:

- **Suboptimality Loss (SL)**: we use (2.16) with no regularization $\mathscr{R}$ and no distance function (i.e., $\kappa = 0$ and $d(\hat{x}^{[i]}, x^{[i]}) = 0$). To avoid the trivial solution $\theta = 0$, we add the constraint $\|\theta\|_1 = 1$. We call this approach SL since it is the result of performing the reformulation steps of Corollary 2.13 using the Suboptimality loss instead of the ASL;

- **ASL-yz**: we use (2.16) with $\theta = (q_y, q_z)$, $s = (A, B, c, 0)$ (A and c have to be aug-

Approach	SL	ASL-z	ASL-yz
Time (seconds)	607	694	7060
Approach	Circumcenter	Cutting plane	Predictability loss
Time (seconds)	5954	981	1336

Table 2.3: Computational time to generate the results of Figure 2.5.

mented to account for the constraints $0 \leq y \leq \mathbb{1}$), $\mathbb{Z}(w) = \{0,1\}^v$, $Q_{yy} = 0$, $\phi_1(w,z) = 1$, $\phi_2(w,z) = z$, $\kappa = 0$, $\Theta = \{\theta \in \mathbb{R}^{u+v} : \theta \geq 0\}$, and $d_z(\hat{z}^{[i]}, z^{[i]}) = \|\hat{z}^{[i]} - z^{[i]}\|_2$.

- **ASL-z**: same as **ASL-yz**, but with $h_k = 0 \; \forall k \in [2u]$ (see discussion in the paragraph after Corollary 2.13);
- **Circumcenter**: we solve (2.21);
- **Cutting plane**: we use the cutting plane algorithm of [Wan09];
- **Predictability loss**: we use the predictability loss of [ASS18] (Appendix 2.A).

Results. Figure 2.5 shows the results for this scenario. The discussions on the interpretations of the results of this section mirror the ones of Figure 2.3 from Section 2.4.1. Once again, the approach based on the ASL outperforms the other approaches on all three performance metrics. In particular, the ASL-z and ASL-yz show similar performance, with the ASL-yz being slightly better overall and when there are very few training examples. However, because the ASL-yz optimization problem has $2N$ times more constraints than the ASL-z optimization problem, it is much more costly to solve, as can be seen from the computational times in Table 2.3.

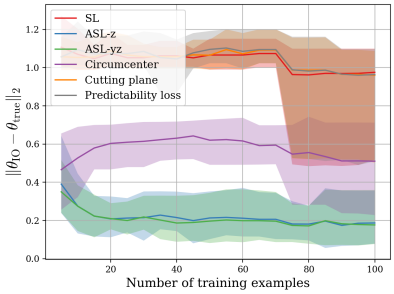
2.4.4. STOCHASTIC APPROXIMATE MIRROR DESCENT

In this section, we numerically evaluate the SAMD algorithm proposed in Section 2.3.

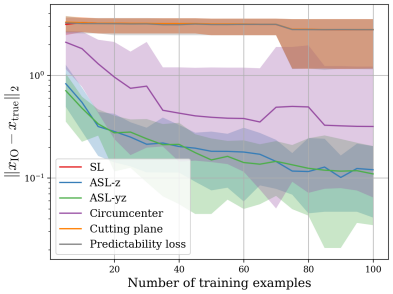
Decision problem of the expert. To generate its decisions, the expert solves the binary linear program (2.20), where $\theta \in \mathbb{R}^n_+$ is the cost vector, $A \in \mathbb{R}^{t \times n}$, $b \in \mathbb{R}^t$, and x is the decision vector. We can write this optimization problem as $\min_{x \in \mathbb{X}(s)} F(s,x)$ by defining $s := (A,b)$, $F(s,x) := \langle \theta, x \rangle$ and $\mathbb{X}(s) := \{x \in \{0,1\}^n : Ax \geq b\}$.

Data generation. To generate training and test data, we sample cost vectors uniformly from $\{\theta \in \mathbb{R}^n : 0 \leq \theta \leq \mathbb{1}\}$, we sample $\hat{A}$ uniformly from $\{A \in \mathbb{R}^{t \times n} : -1 \leq A \leq 0\}$ and $\hat{b}$ uniformly from $\{b \in \mathbb{R}^t : -\frac{n}{3} \leq b \leq 0\}$. To make sure the problem instances are feasible, we checked if the sum of each row of $\hat{A}$ is larger than the respective component of $\hat{b}$. Given a signal $\hat{s}$, we generate a response $\hat{x}$ by solving (2.20), i.e., $\hat{x} \in \operatorname{argmin}_{x \in \mathbb{X}(\hat{s})} \langle \theta, x \rangle$. To evaluate the IO approaches, we generate 10 random cost vectors θ_{true}. For each of these cost vectors, we generate a training dataset $\widehat{\mathscr{D}}_{\text{train}} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$ and a test dataset $\widehat{\mathscr{D}}_{\text{test}} = \{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^N$, with $N = 50$, $n = 20$ and $t = 15$.

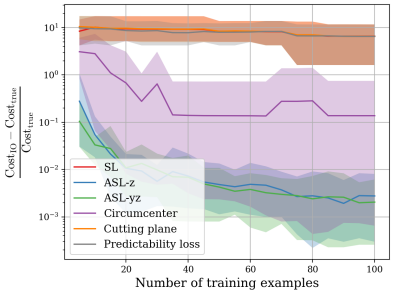
IO approaches. We tackle this IO problem using (2.10) with $\kappa = 0.01$, $\mathscr{R}(\theta) = \|\theta\|_1$, $\Theta = \{\theta \in \mathbb{R}^n : \theta \geq 0\}$ and the ASL with $s = (A,b)$, $\mathbb{X}(s) = \{x \in \{0,1\}^n : Ax \leq b\}$, $\phi(s,x) = x$ and $d(\hat{x},x) = \|x - \hat{x}\|_1$. When testing algorithms with exponentiated updates (i.e., mirror descent updates with $\omega(\theta) = \sum_{i=1}^n \theta_i \log(\theta_i)$), we solve the reformulation (2.17), with $\tilde{\kappa}$



(a) Difference between the true cost vector (θ_{true}) and the one learned using IO (θ_{IO}).



(b) Average error between the decision generated by θ_{true} and θ_{IO}.



(c) Relative difference between the cost of the decisions generated using θ_{true} and θ_{IO}.

Figure 2.5: Out-of-sample results for the mixed-integer feasible set scenario.

chosen such that (2.10) and (2.17) have the same optimal solution. We compare eight algorithms, which result from all possible combinations of standard or mirror descent updates, deterministic or stochastic subgradients, and exact or approximate subgradients.

(i) **Subgradient method (SM)**: Algorithm 1 with $\omega(\theta) = \frac{1}{2}\|\theta\|_2^2$ and exact subgradients computed using the entire dataset;

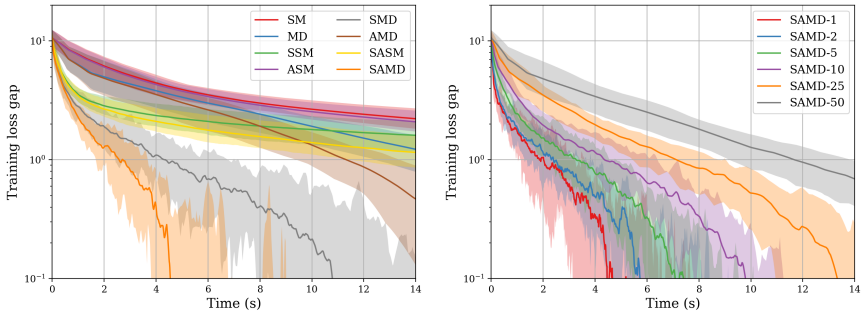
(ii) **Mirror descent (MD)** with exponentiated updates: Algorithm 1 with exact subgradients computed using the entire dataset and $\omega(\theta) = \sum_{i=1}^{n} \theta_i \log(\theta_i)$;

(iii) **Stochastic subgradient method (SSM)**: Algorithm 1 with $\omega(\theta) = \frac{1}{2}\|\theta\|_2^2$ and exact stochastic subgradients;

(iv) **Approximated subgradient method (ASM)**: Algorithm 1 with $\omega(\theta) = \frac{1}{2}\|\theta\|_2^2$ and approximate subgradients computed using the entire dataset;

(v) **Stochastic mirror descent (SMD)** with exponentiated updates: Algorithm 1 with $\omega(\theta) = \sum_{i=1}^{n} \theta_i \log(\theta_i)$ and exact stochastic subgradients;

(vi) **Approximate mirror descent (AMD)** with exponentiated updates: Algorithm 1 with



(a) Convergence of the first-order optimization algorithms.

(b) Convergence of the SAMD algorithms for different sizes of batch sizes.

Figure 2.6: Convergence time of different first-order optimization algorithms and for different batch sizes.

$\omega(\theta) = \sum_{i=1}^{n} \theta_i \log(\theta_i)$ and and approximate subgradients computed using the entire dataset;

(vii) **Stochastic approximate subgradient method (SASM)**: Algorithm 1 with approximate stochastic subgradients and $\omega(\theta) = \frac{1}{2}\|\theta\|_2^2$.

(viii) **Stochastic Approximate Mirror Descent (SAMD)** with exponentiated updates, i.e., Algorithm 1 with $\omega(\theta) = \sum_{i=1}^{n} \theta_i \log(\theta_i)$ and approximate stochastic subgradients.

For all algorithms, we use $\eta_t = 1/(\|\tilde{g}_{\varepsilon_t}(\theta_t)\|_* \sqrt{t})$. To compute approximate subgradients for the ASM and SAMD algorithms, we give the solver (in our case, Gurobi) a time limit of 0.03 seconds to solve the optimization problem in line 4 of Algorithm 1. If the solver is not able to find an optimal solution within this time limit, it returns the best feasible solution found.

Results. To compare the performance of the algorithms, we report their results in terms of running time instead of the number of iterations. Figure 2.6a shows the convergence of the proposed algorithms in terms of the training loss gap. More precisely, defining $f(\theta) := \kappa\mathscr{R}(\theta) + \frac{1}{N}\sum_{i=1}^{N} \ell_\theta(\hat{s}^{[i]}, \hat{x}^{[i]})$, the training loss gap of some θ_t is defined as $f(\theta_t) - \min_{\theta\in\Theta} f(\theta)$. From this plot, it is clear that each modification of the standard subgradient method (i.e., mirror descent updates, stochastic subgradients, and approximate subgradients) contributes to improving the convergence speed of the algorithms, with the fastest convergence achieved by the combination of all of these improvements, i.e., the SAMD algorithm. In Figure 2.6b, we show the convergence of the SAMD using different batch sizes to compute stochastic subgradients. That is, instead of using only one sampled example at each iteration (line 3 of Algorithm 1), we sample a batch of B examples and use this batch of data to compute the stochastic subgradient, which we call SAMD-B. From Figure 2.6b we can see that even though smaller batches lead to more variance in the converge of the algorithm (as expected) it also leads to a faster empirical convergence for this experiment.

2.A. THEORETICAL PROPERTIES OF THE AUGMENTED SUBOPTIMALITY LOSS

Define $F_{\theta,\phi}(s,x) := \langle \theta, \phi(s,x) \rangle$. In this section, we discuss several theoretical properties of the Augmented Suboptimality Loss (ASL), and show how it can be interpreted as a convex surrogate of the *Generalized Predictability Loss* (GPL). The GPL is a natural loss function for IO problems and is defined as follows.

Definition 2.19 (Generalized Predictability Loss)**.** *Given a signal-response pair* $(\hat{s}, \hat{x})$*, the Generalized Predictability Loss of a cost vector* θ *is given by*

$$\begin{aligned} \ell_\theta^{\text{pred}}(\hat{s}, \hat{x}) := \min_x \; & d(\hat{x}, x) \\ \text{s.t.} \; & x \in \operatorname*{argmin}_{y \in \mathbb{X}(\hat{s})} F_{\theta,\phi}(\hat{s}, y), \end{aligned} \tag{2.24}$$

where $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}_+$ *is a distance function.*

The GPL has an intuitive interpretation for IO problems: given a signal s, it computes how close (in terms of d) are $\hat{x}$ (the expert's response) and x (an optimal response according to $F_{\theta,\phi}$). Consequently, by solving (2.10) using the GPL, we optimize for a cost $F_{\theta,\phi}$ that best reproduces the responses taken by the expert for each signal s. For the special case when $d(x,y) = \|x - y\|_2^2$, the GPL reduces to the so-called *predictability loss*. Unfortunately, when using the GPL, problem (2.10) is an NP-hard bi-level optimization problem in general [ASS18].

To come up with losses that are convex w.r.t. θ, and still meaningful for IO problems, we can use the concept of *convex surrogate functions*. Informally, given a nonconvex function g, a convex surrogate function h is a convex upper bound of g. The idea is then to minimize h instead of g. The hope is that by minimizing h, we also minimize (at least to some extent) the original nonconvex function g. Interestingly, we can show that the ASL is a convex surrogate for the GPL and that it possesses several properties attractive for IO loss functions. In particular, the properties of the ASL shown in Proposition 2.21 are similar to (and in some sense generalizations of) the properties of the Suboptimality Loss presented in [Moh+18a].

Assumption 2.20 (Optimizer condition)**.** *Let the distance function* $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}_+$ *be such that* $d(x,y) = 0$ *if and only if* $x = y$*. Then,* $\forall\, x \in \mathbb{X}(s)$ *and* $\forall\, s \in \mathbb{S}$*, there exists a* $\theta \in \Theta$ *such that*

$$y^\star \in \operatorname*{argmin}_{y \in \mathbb{X}(s)} F_{\theta,\phi}(s,y) \implies F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y^\star) \geq d(x,y^\star).$$

Assumption 2.20 can be interpreted as a "unique minimizer" condition and it is necessary to prove the "consistency" property of Proposition 2.21. This is a strong assumption and does not hold for general IO problems. However, this assumption is not critical for the algorithm design and analysis presented in this chapter. Moreover, empirically, IO algorithms based on the ASL perform well even for IO problems that do not possess the "unique minimizer" property (for example, see the numerical results of Section 2.4). Finally, we note that Assumption 2.20 can be interpreted as a generalization of other assumptions from the literature, which are also used to prove consistency-like

properties of loss functions. We list three such examples, where we use the notation $y^\star \in \operatorname{argmin}_{y\in\mathbb{X}(s)} F_{\theta,\phi}(s,y)$.

- Let $d(x,y) = I(x,y)$, the 0-1 distance. In this case, Assumption 2.20 reads as

$$F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y^\star) \geq 1 \quad \forall (s,x) \in \mathbb{S} \times \mathbb{X}(s) \setminus \{y^\star\}.$$

  The inequality above holds, for instance, in the case $F_{\theta,\phi}(s,x) = \langle \theta, x \rangle$ with $\theta \in \mathbb{Z}_+^n$ (a vector of positive integers) and $\mathbb{X}(x) = \{0,1\}^n$ [BPS17, Corollary 3.6].

- Let $d(x,y) = \|x-y\|_2^2$. In this case, Assumption 2.20 reads as

$$F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y^\star) \geq \|x-y^\star\|_2^2, \quad \forall (s,x) \in \mathbb{S} \times \mathbb{X}(s).$$

  Let the set $\mathbb{X}(s)$ be μ-strongly convex with $\mu = 2/\|\nabla_x F_{\theta,\phi}(s,y^\star)\|_2$. Then, one can show that Assumption 2.20 holds [El +19, Proposition 1]. Particularly, notice that the inequality holds even when $F_{\theta,\phi}(s,x) = \langle \theta, x \rangle$.

- Let $d(x,y) = \|x-y\|_2^2$. In this case, Assumption 2.20 reads as

$$F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y^\star) \geq \|x-y^\star\|_2^2, \quad \forall (s,x) \in \mathbb{S} \times \mathbb{X}(s).$$

  One can show that this inequality holds when $F_{\theta,\phi}(s,x)$ is 2-strongly convex w.r.t. x. Thus, one can interpret Assumption 2.20 as a generalization of the strong-convexity assumption used in [Moh+18a, Proposition 2.5].

Proposition 2.21 (Properties of the ASL)**.** *The ASL ℓ_θ has the following properties:*

- *(**Convex surrogate**) $\ell_\theta^{\text{pred}}(s,x) \leq \ell_\theta(s,x) \quad \forall (s,x) \in \mathbb{S} \times \mathbb{X}(s)$;*
- *(**Convexity**) ℓ_θ is convex w.r.t. θ;*
- *(**Nonnegativity**) $\ell_\theta(s,x) \geq 0, \quad \forall (s,x) \in \mathbb{S} \times \mathbb{X}(s)$;*
- *(**Consistency**) $\ell_\theta(s,x) = 0 \implies x \in \operatorname{argmin}_{y\in\mathbb{X}(s)} F_{\theta,\phi}(s,y), \quad \forall s \in \mathbb{S}$. If Assumption 2.20 holds, then $\ell_\theta(s,x) = 0 \impliedby x \in \operatorname{argmin}_{y\in\mathbb{X}(s)} F_{\theta,\phi}(s,y), \quad \forall s \in \mathbb{S}$.*

Proof. Let $y^\star \in \operatorname{argmin}_{y\in\mathbb{X}(s)} F_{\theta,\phi}(s,y)$.

(Convex surrogate) By the definition of the generalized predictability loss and $y^\star$, we have $\ell_\theta^{\text{pred}}(s,x) \leq d(x,y^\star)$ and

$$F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y^\star) \geq 0, \tag{2.25}$$

for all for all $x \in \mathbb{X}(s)$, for all $s \in \mathbb{S}$. Thus,

$$\begin{aligned} \ell_\theta^{\text{pred}}(s,x) &\leq d(x,y^\star) \\ &\leq d(x,y^\star) + F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y^\star) && \text{(Eq. (2.25))} \\ &\leq \max_{y\in\mathbb{X}(s)} \left\{ d(x,y) + F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y) \right\} = \ell_\theta(s,x). \end{aligned}$$

(Convexity) Using the fact that $F_{\theta,\phi}(s,x) = \langle \theta, \phi(s,x) \rangle$, we conclude that $\ell_\theta(s,x)$ is the pointwise maximum of linear functions of θ, which is convex w.r.t. θ.

(Nonnegative) Since $\ell_\theta^{\text{pred}}(s,x)$ is nonnegative, and we have shown that the ASL upper bounds the GPL, this implies $\ell_\theta(s,x) \geq 0$.

(Consistency $\Longrightarrow$)

$$\begin{aligned}
\ell_\theta(s,x) = 0 &\Longrightarrow \max_{y \in \mathbb{X}(s)} \left\{ F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y) + d(x,y) \right\} = 0 \\
&\Longrightarrow F_{\theta,\phi}(s,x) = \min_{y \in \mathbb{X}(s)} \left\{ F_{\theta,\phi}(s,y) - d(x,y) \right\} \\
&\Longrightarrow F_{\theta,\phi}(s,x) \leq F_{\theta,\phi}(s,y) - d(x,y), \quad \text{for any } y \in \mathbb{X}(s) \\
&\Longrightarrow F_{\theta,\phi}(s,x) \leq F_{\theta,\phi}(s,y^\star) - d(x,y^\star) \\
&\Longrightarrow F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y^\star) \leq 0, \qquad (2.26)
\end{aligned}$$

where the last inequality follows from the facts that $d(x,y^\star)$ is nonnegative. From (2.25) and (2.26) and the definition of $y^\star$, we conclude $x \in \operatorname{argmin}_{y \in \mathbb{X}(s)} F_{\theta,\phi}(s,y)$.

(Consistency $\Longleftarrow$) By Assumption 2.20, we have that

$$\begin{aligned}
x \in \operatorname*{argmin}_{y \in \mathbb{X}(s)} F_{\theta,\phi}(s,y) &\Longrightarrow F_{\theta,\phi}(s,y) - F_{\theta,\phi}(s,x) \geq d(y,x), \quad \text{for any } y \in \mathbb{X}(s) \\
&\Longrightarrow F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y) + d(y,x) \leq 0, \quad \text{for any } y \in \mathbb{X}(s) \\
&\Longrightarrow \max_{y \in \mathbb{X}(s)} \left\{ F_{\theta,\phi}(s,x) - F_{\theta,\phi}(s,y) + d(y,x) \right\} \leq 0 \Longrightarrow \ell_\theta(s,x) \leq 0.
\end{aligned}$$

Since we have already shown that $\ell_\theta(s,x) \geq 0$, this implies $\ell_\theta(s,x) = 0$. □

2.B. CONTINUOUS PROBLEMS: SPECIAL CASES

A key step in the derivation of the reformulation presented in Theorem 2.12 is dualizing the maximization problem in the ASL with respect to the continuous part of the decision vector. For the reformulation to be exact, strong duality needs to hold, i.e., the maximization problem needs to be concave w.r.t. the continuous variable. As mentioned in Section 2.2.2, using a general ∞-norm penalization comes at the expense of increasing the number of constraints in the reformulation by a factor of $2u$. However, for some special cases, one can dualize the distance-augmented problem without increasing the number of constraints of the final problem. Here we briefly discuss two such examples: *linear programs with totally unimodular constraint matrix* and *quadratically constrained quadratic programs*.

LPs with totally unimodular constraint matrix. Let us consider IO problems with linear hypothesis $\langle \theta, \phi(s,x) \rangle = \langle \theta, x \rangle$ and linear constraints $Ax \leq b, 0 \leq x \leq \mathbb{1}$. For this problem, the inner maximization problem of the ASL with $d(\hat{x},x) = \|\hat{x} - x\|_1$ is of the form

$$\begin{aligned}
\max_{x \in \mathbb{R}^n} \quad & \langle \theta, \hat{x} - x \rangle + \|\hat{x} - x\|_1 \\
\text{s.t.} \quad & Ax \leq b \qquad (2.27) \\
& 0 \leq x \leq \mathbb{1}.
\end{aligned}$$

Due to the norm in the objective function, this is a nonconcave optimization problem in general. However, consider the case when the constraint matrix A in (2.27) is *totally*

unimodular and b is an integer vector. Totally unimodular constraint matrices appear in many classical optimization problems, for instance, shortest path problems, bipartite graph matching, and maximum flow problems [CCZ14]. In this case, it is a well-known result that every extreme point of the polyhedron $\{x : Ax \le b,\ 0 \le x \le \mathbb{1}\}$ is integral [HK10], thus, (2.27) has integral (in this particular case, binary) optima. Combined with the fact that, if $\hat{x}$ and x are binary vectors, we have that $\|\hat{x} - x\|_1 = \langle \mathbb{1} - 2\hat{x}, x\rangle + \langle \mathbb{1}, \hat{x}\rangle$, the optimization problem (2.27) is equivalent to

$$\begin{aligned} \max_{x\in\mathbb{R}^n} \quad & \langle \theta, \hat{x} - x\rangle + \langle \mathbb{1} - 2\hat{x}, x\rangle + \langle \mathbb{1}, \hat{x}\rangle \\ \text{s.t.} \quad & Ax \le b \\ & 0 \le x \le \mathbb{1}, \end{aligned}$$

that is, problem (2.27) is equivalent to an LP and, thus, can be dualized without increasing the number of constraints in the final IO reformulation.

Quadratically constrained quadratic programs. Next, consider IO problems with quadratic hypothesis $\langle \theta, \phi(s,x)\rangle = \langle x, Qx\rangle + 2\langle x, q\rangle$ and one quadratic constraint $\langle x, Ax\rangle + 2\langle x, b\rangle + c \le 0$. For this problem, the inner maximization problem of the ASL with $d(\hat{x}, x) = \|\hat{x} - x\|_2^2$ is of the form

$$\begin{aligned} \max_{x\in\mathbb{R}^n} \quad & \langle \hat{x}, Q\hat{x}\rangle + 2\langle \hat{x}, q\rangle - \langle x, Qx\rangle - 2\langle x, q\rangle + \|\hat{x} - x\|_2^2 \\ \text{s.t.} \quad & \langle x, Ax\rangle + 2\langle x, b\rangle + c \le 0. \end{aligned} \tag{2.28}$$

Problem (2.28) is sometimes called a *Generalized Trust Region Problem*, and has numerous applications to, for example, robust optimization, signal processing, and compressed sensing [Mor93; PW14; WK22]. Moreover, (2.28) is not a concave maximization problem unless $Q - I \succcurlyeq 0$. However, it is a well-known result that strong duality holds for (2.28) and its dual program is

$$\begin{aligned} \min_{t,\lambda} \quad & \langle \hat{x}, Q\hat{x}\rangle + 2\langle \hat{x}, q\rangle + \langle \hat{x}, \hat{x}\rangle - t \\ \text{s.t.} \quad & \lambda \ge 0 \\ & \begin{bmatrix} Q - I + \lambda A & q + \hat{x} + \lambda b \\ * & \lambda c - t \end{bmatrix} \succcurlyeq 0, \end{aligned}$$

provided Slater's constraint qualification is satisfied, i.e., there exists an x with $\langle x, Ax\rangle + \langle x, b\rangle + c \le 0$ [BV04, Appendix B].

2.C. PROOFS

2.C.1. PROOF OF THEOREM 2.2

Using the facts that $\|\tilde{\theta}\|_2 = \|\theta\|_2 = 1$, the range of the arccos is $[0, \pi]$, $-\cos(\gamma)$ is monotone increasing for $\gamma \in [0, \pi]$, and $-\langle \tilde{\theta}, \theta\rangle = \frac{1}{2}\|\tilde{\theta} - \theta\|_2^2 - 1$, evaluating the cost function of (2.3) is equivalent to

$$\begin{aligned} \max_{\tilde{\theta}} \quad & \|\theta - \tilde{\theta}\|_2^2 \\ \text{s.t.} \quad & \tilde{\theta} \in \mathbb{C}, \quad \|\tilde{\theta}\|_2 = 1. \end{aligned} \tag{2.29}$$

Next, applying the change of variables $\tilde{\theta} \mapsto R\tilde{\theta}$, where $R \in \mathbb{R}^{n\times n}$ is an orthonormal matrix with its first column equal to θ, (2.29) is equivalent to

$$\begin{array}{ll} \max\limits_{\tilde{\theta}} & \|e_1 - \tilde{\theta}\|_2^2 \\ \text{s.t.} & R\tilde{\theta} \in \mathbb{C}, \quad \|\tilde{\theta}\|_2 = 1. \end{array} \tag{2.30}$$

Then, defining $\mathbb{C} := \{R\tilde{\theta} = (x, y) \in \mathbb{R}\times\mathbb{R}^{n-1} : a_i x + \langle b_i, y\rangle \le 0 \ \forall i \in [N],\ -x + \langle e_j, y\rangle \le 0$ and $-x - \langle e_j, y\rangle \le 0 \ \forall j \in [n-1]\}$, we can rewrite (2.30) as

$$\begin{array}{lll} \max\limits_{x,y} & (x-1)^2 + \|y\|_2^2 & \\ \text{s.t.} & a_i x + \langle b_i, y\rangle \le 0 & \forall i \in [N] \\ & -x + \langle e_j, y\rangle \le 0 & \forall j \in [n-1] \\ & -x - \langle e_j, y\rangle \le 0 & \forall j \in [n-1] \\ & x^2 + \|y\|_2^2 = 1. & \end{array} \tag{2.31}$$

Next, we change the objective function of (2.31) from $f(x, y) = (x-1)^2 + \|y\|_2^2$ to $g(x, y) = \|(1/x)y\|_2^2$. The resulting optimization problem is equivalent to (2.31) because g is a strictly positive monotonic transformation of f, that is, for any feasible (x_1, y_1) and (x_2, y_2), we have that

$$\begin{aligned} & f(x_1, y_1) < f(x_2, y_2) \\ \iff & (x_1-1)^2 + \|y_1\|_2^2 < (x_2-1)^2 + \|y_2\|_2^2 \\ \iff & -2x_1 < -2x_2 \\ \iff & \frac{1}{x_1^2} < \frac{1}{x_2^2} \\ \iff & \frac{1-x_1^2}{x_1^2} < \frac{1-x_2^2}{x_2^2} \\ \iff & \left\|\frac{1}{x_1}y_1\right\|_2^2 < \left\|\frac{1}{x_2}y_2\right\|_2^2 \\ \iff & g(x_1, y_1) < g(x_2, y_2), \end{aligned}$$

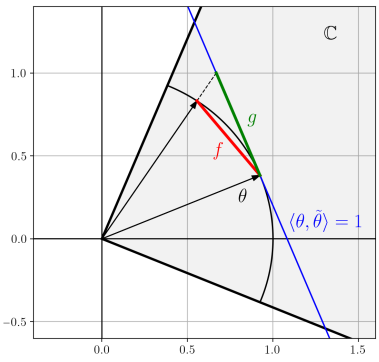


Illustration of the monotonicity of the transformation from f to g.

where we used the fact that from the inequalities $-x \pm \langle e_j, y\rangle \le 0$ and the equality $x^2 + \|y\|_2^2 = 1$, any feasible x is strictly larger than 0. Thus, (2.31) is equivalent to

$$\begin{array}{lll} \max\limits_{x,y,z} & \|z\|_2^2 & \\ \text{s.t.} & a_i x + \langle b_i, y\rangle \le 0 & \forall i \in [N] \\ & -x + \langle e_j, y\rangle \le 0 & \forall j \in [n-1] \\ & -x - \langle e_j, y\rangle \le 0 & \forall j \in [n-1] \\ & x^2 + \|y\|_2^2 = 1, \quad z = \dfrac{1}{x}y. & \end{array} \tag{2.32}$$

Next, substituting $y = xz$ in the inequality constraints, and using the fact that $x > 0$ to rearrange the constraints, we arrive at

$$\begin{aligned} \max_{z} \quad & \|z\|_2^2 \\ \text{s.t.} \quad & \langle b_i, z\rangle \le -a_i \quad \forall i \in [N] \\ & -1 \le z \le 1, \end{aligned} \tag{2.33}$$

where we dropped the equality constraints $x = 1/\sqrt{1+\|z\|_2^2}$ and $y = \left(1/\sqrt{1+\|z\|_2^2}\right) z$, since they are the only ones that depend on x and y. Therefore, we have shown that evaluating the cost function of the circumcenter problem is equivalent to a quadratic maximization problem over a polytope, which is NP-hard [Sah74].

2.C.2. PROOF OF THEOREM 2.5

First, the non-zero constraints can be enforced via $\|\theta\|_2 = \|\tilde{\theta}\|_2 = 1$, and similar to the proof of Theorem 2.2, one can show that

$$\operatorname*{argmax}_{\|\theta\|_2=1} \min_{\substack{\tilde{\theta}\in\overline{\operatorname{int}(\mathbb{C})} \\ \|\tilde{\theta}\|_2=1}} a(\theta,\tilde{\theta}) = \operatorname*{argmax}_{\|\theta\|_2=1} \min_{\substack{\tilde{\theta}\in\overline{\operatorname{int}(\mathbb{C})} \\ \|\tilde{\theta}\|_2=1}} \|\theta-\tilde{\theta}\|_2^2.$$

Since for any $\theta \in \overline{\operatorname{int}(\mathbb{C})}$, we can always choose $\tilde{\theta} = \theta$ and have $\|\theta - \tilde{\theta}\|_2^2 = 0$, the θ that maximizes this minimax problem will always be in $\mathbb{C}$. Consequently, to minimize the distance to any $\theta \in \mathbb{C}$, we have that the optimal $\tilde{\theta}$ will always be in the boundary of $\mathbb{C}$. Using these observations, one can show the following equivalence for the inner minimization problem:

$$\min_{\substack{\tilde{\theta}\in\overline{\operatorname{int}(\mathbb{C})} \\ \|\tilde{\theta}\|_2=1}} \|\theta-\tilde{\theta}\|_2^2 = \min_{\substack{x^{[i]}\in\mathbb{X}(\hat{s}^{[i]}) \\ i\in[N]}} \left\{ \min_{\substack{\langle\tilde{\theta},\phi(\hat{s}^{[i]},x^{[i]})-\phi(\hat{s}^{[i]},\hat{x}^{[i]})\rangle=0 \\ \|\tilde{\theta}\|_2=1}} \|\theta-\tilde{\theta}\|_2^2 \right\}, \tag{2.34}$$

which follows from the fact that the optimal $\tilde{\theta}$ being in boundary of $\mathbb{C}$ implies $\langle\tilde{\theta}, \phi(\hat{s}^{[i]}, x^{[i]}) - \phi(\hat{s}^{[i]}, \hat{x}^{[i]})\rangle = 0$ for some $i \in [N]$ and $x^{[i]} \in \mathbb{X}(\hat{s}^{[i]})$ (see Definition 1.2).

Next, we derive a closed-form solution to the minimization problems inside curly brackets in (2.34). Namely, we show that

$$\operatorname*{argmin}_{\substack{\langle\tilde{\theta},w\rangle=0 \\ \|\tilde{\theta}\|_2=1}} \|\theta-\tilde{\theta}\|_2^2 = \frac{p(\theta,w)}{\|p(\theta,w)\|_2}, \tag{2.35}$$

where

$$p(\theta,w) := \operatorname*{argmin}_{\langle\tilde{\theta},w\rangle=0} \|\theta-\tilde{\theta}\|_2^2 = \theta - \frac{\langle\theta,w\rangle}{\|w\|_2^2} w \tag{2.36}$$

is the euclidean projection of θ onto the hyperplane defined by w. We prove (2.35) by contradiction. Namely, we show that if there exists a vector α with lower cost value

than $p(\theta,w)/\|p(\theta,w)\|_2$ for (2.35), this contradicts the optimality of $p(\theta,w)$ for problem (2.36). Thus, let $\alpha \in \mathbb{R}^p$ be such that $\langle \alpha, w\rangle = 0$ and $\|\alpha\|_2 = 1$, and assume

$$\left\| \theta - \frac{p(\theta,w)}{\|p(\theta,w)\|_2} \right\|_2^2 > \|\theta - \alpha\|_2^2.$$

Consequently, we have that

$$\begin{aligned}\left\| \theta - \frac{p(\theta,w)}{\|p(\theta,w)\|_2} \right\|_2^2 > \|\theta-\alpha\|_2^2 &\iff 1+1-\frac{2}{\|p(\theta,w)\|_2}\langle \theta, p(\theta,w)\rangle > 1+1-2\langle\theta,\alpha\rangle \\ &\iff -2\langle\theta,p(\theta,w)\rangle > -2\|p(\theta,w)\|_2\langle\theta,\alpha\rangle,\end{aligned}$$

which follows by expanding the norm squared and the facts that $\|\theta\|_2 = \|\alpha\|_2 = 1$. Adding $\|\theta\|_2^2 + \|p(\theta,w)\|_2^2$ to both sides and using the identity $\|\theta - p(\theta,w)\|_2^2 = \|\theta\|_2^2 + \|p(\theta,w)\|_2^2 - 2\langle\theta, p(\theta,w)\rangle$, we get that

$$\left\| \theta - \frac{p(\theta,w)}{\|p(\theta,w)\|_2} \right\|_2^2 > \|\theta-\alpha\|_2^2 \iff \|\theta - p(\theta,w)\|_2^2 > \|\theta\|_2^2 + \|p(\theta,w)\|_2^2 - 2\|p(\theta,w)\|_2\langle\theta,\alpha\rangle.$$

Next, define $\beta := \|p(\theta,w)\|_2\alpha$, and notice that $\langle\beta, w\rangle = 0$ and $\|\beta\|_2^2 = \|p(\theta,w)\|_2^2$. Thus, we arrive at

$$\left\| \theta - \frac{p(\theta,w)}{\|p(\theta,w)\|_2} \right\|_2^2 > \|\theta-\alpha\|_2^2 \iff \|\theta - p(\theta,w)\|_2^2 > \|\theta\|_2^2 + \|\beta\|_2^2 - 2\langle\theta,\beta\rangle = \|\theta-\beta\|_2^2.$$

Therefore, since $\|\theta - p(\theta,w)\|_2^2 > \|\theta-\beta\|_2^2$ contradicts the optimality of $p(\theta,w)$ in (2.36), (2.35) follows by contradiction.

Combining (2.34) and (2.35), we have that

$$\min_{\substack{\tilde\theta \in \overline{\text{int}(\mathbb{C})} \\ \|\tilde\theta\|_2 = 1}} \|\theta - \tilde\theta\|_2^2 = \min_{\substack{x^{[i]} \in \mathbb{X}(\hat s^{[i]}) \\ i\in[N]}} \left\| \theta - \frac{p(\theta, \phi(\hat s^{[i]}, x^{[i]}) - \phi(\hat s^{[i]}, \hat x^{[i]}))}{\|p(\theta, \phi(\hat s^{[i]}, x^{[i]}) - \phi(\hat s^{[i]}, \hat x^{[i]}))\|_2} \right\|_2^2.$$

To conclude the proof, notice that

$$\begin{aligned}\operatorname*{argmin}_w \left\| \theta - \frac{p(\theta,w)}{\|p(\theta,w)\|_2} \right\|_2^2 &= \operatorname*{argmin}_w\ a\left(\theta, \frac{p(\theta,w)}{\|p(\theta,w)\|_2}\right) && \text{(2.37a)} \\ &= \operatorname*{argmin}_w\ a\big(\theta, p(\theta,w)\big) \\ &= \operatorname*{argmin}_w\ \sin\big(a(\theta,p(\theta,w))\big) && \text{(2.37b)} \\ &= \operatorname*{argmin}_w\ \big\|\theta - p(\theta,w)\big\|_2^2 && \text{(2.37c)} \\ &= \operatorname*{argmin}_w\ \frac{|\langle\theta,w\rangle|}{\|w\|_2}, && \text{(2.37d)}\end{aligned}$$

where (2.37a) follows from the definition of angle, (2.37b) follows from $a(\theta,p(\theta,w)) \le \pi/2$, (2.37c) follows from $p(\theta,w) \perp \theta - p(\theta,w)$ and (2.37d) follows from Eq. (2.36). Putting

it together, we conclude that

$$\theta^{\text{in}} \in \operatorname*{argmax}_{\|\theta\|_2=1} \min_{\substack{\tilde{\theta}\in\text{int}(\mathbb{C}) \\ \|\tilde{\theta}\|_2=1}} a(\theta,\tilde{\theta}) = \operatorname*{argmax}_{\|\theta\|_2=1} \min_{x^{[i]}\in\mathbb{X}(\hat{s}^{[i]}),i\in[N]} \frac{|\langle\theta,\phi(\hat{s}^{[i]},x^{[i]})-\phi(\hat{s}^{[i]},\hat{x}^{[i]})\rangle|}{\|\phi(\hat{s}^{[i]},x^{[i]})-\phi(\hat{s}^{[i]},\hat{x}^{[i]})\|_2}.$$

Applying an epigraph reformulation and rearranging the resulting inequality constraints ends the proof.

2.C.3. PROOF OF COROLLARY 2.6

First, we show that the assumption that $\text{int}(\mathbb{C}) \neq \emptyset$ implies that the optimal r of problem (2.6) is positive. To see this, notice that $\text{int}(\mathbb{C}) \neq \emptyset$ implies that there exists some $\theta \in \mathbb{R}^p$ such that $\langle\theta,\phi(\hat{s}^{[i]},x^{[i]})-\phi(\hat{s}^{[i]},\hat{x}^{[i]})\rangle > 0$, $\forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]})$, $\forall i \in [N]$, which follows Definition 1.2. Next, simply notice that

$$r = \frac{\min_{x^{[i]}\in\mathbb{X}(\hat{s}^{[i]}),i\in[N]} \langle\theta,\phi(\hat{s}^{[i]},x^{[i]})-\phi(\hat{s}^{[i]},\hat{x}^{[i]})\rangle}{\max_{x^{[i]}\in\mathbb{X}(\hat{s}^{[i]}),i\in[N]} \|\phi(\hat{s}^{[i]},x^{[i]})-\phi(\hat{s}^{[i]},\hat{x}^{[i]})\|_2} > 0$$

is a feasible r for problem (2.6), which shows that the optimal $r > 0$. Thus, problem (2.6) can be written as

$$\begin{aligned}
\min_{\theta,r}\quad & \|\theta\|_2/r \\
\text{s.t.}\quad & \langle\theta,\phi(\hat{s}^{[i]},\hat{x}^{[i]})-\phi(\hat{s}^{[i]},x^{[i]})\rangle + r\|\phi(\hat{s}^{[i]},x^{[i]})-\phi(\hat{s}^{[i]},\hat{x}^{[i]})\|_2 \le 0 \qquad \forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]}),\ \forall i \in [N] \\
& \|\theta\|_2 = 1,
\end{aligned}$$

where we used the facts that $\|\theta\|_2 = 1$ and the optimal $r > 0$. Next, applying the change of variables $\theta/r \to \bar{\theta}$, we get

$$\begin{aligned}
\min_{\bar{\theta},r}\quad & \|\bar{\theta}\|_2 \\
\text{s.t.}\quad & \langle\bar{\theta},\phi(\hat{s}^{[i]},\hat{x}^{[i]})-\phi(\hat{s}^{[i]},x^{[i]})\rangle + \|\phi(\hat{s}^{[i]},x^{[i]})-\phi(\hat{s}^{[i]},\hat{x}^{[i]})\|_2 \le 0 \qquad \forall x^{[i]} \in \mathbb{X}(\hat{s}^{[i]}),\ \forall i \in [N] \\
& \|\bar{\theta}\|_2 = 1/r.
\end{aligned}$$

Finally, notice that since the optimization variable r only appears in the constraint $\|\bar{\theta}\|_2 = 1/r$, and for any $\bar{\theta}$ we can set $r = 1/\|\bar{\theta}\|_2$, the optimization variable r does not affect the optimal value of $\bar{\theta}$, and we can simply drop the constraint $\|\bar{\theta}\|_2 = 1/r$.

2.C.4. PROOF OF THEOREM 2.12

For the feasible set (2.13) and hypothesis function (2.14), the ASL with $d(\hat{x},x) = \|\hat{y}-y\|_\infty + d_z(\hat{z},z)$ can be written as

$$\begin{aligned}
\max_{\substack{z\in\mathbb{Z}(\hat{w}) \\ h\in\mathbb{H}}} \max_{y}\quad & \langle\theta,\phi(\hat{s},\hat{x})\rangle - \langle y,Q_{yy}y\rangle - \langle y,Q\phi_1(\hat{w},z)\rangle - \langle q,\phi_2(\hat{w},z)\rangle + \langle h,\hat{y}-y\rangle + d_z(\hat{z},z) \\
\text{s.t.}\quad & \hat{A}y + \hat{B}z \le \hat{c},
\end{aligned}$$

where we use the identity $\|\hat{y}-y\|_\infty = \max_{h\in\mathbb{H}}\langle h, \hat{y}-y\rangle$ with $\mathbb{H} := \{h \in \{-1,0,1\}^u : \|h\|_1 = 1\}$. Since we assume $Q_{yy} \succcurlyeq 0$, the inner maximization problem is convex, and by strong duality, we can write its dual as

$$\begin{aligned}\max_{\substack{z\in\mathbb{Z}(\hat{w})\\ h\in\mathbb{H}}}\min_{\alpha,\lambda}\quad & \langle\theta,\phi(\hat{s},\hat{x})\rangle + \alpha + \langle\lambda, \hat{c}-\hat{B}z\rangle - \langle q,\phi_2(\hat{w},z)\rangle + \langle h,\hat{y}\rangle + d_z(\hat{z},z)\\ \text{s.t.}\quad & \|Q\phi_1(\hat{w},z)+h+\hat{A}^\top\lambda\|^2_{Q^\dagger_{yy}} \le 4\alpha\\ & Q\phi_1(\hat{w},z)+h+\hat{A}^\top\lambda \in \mathscr{R}(Q_{yy})\\ & \lambda \ge 0.\end{aligned}$$

Next, applying a Schur complement transformation (e.g., [BV04, Section A.5.5]) to the constraints of the optimization problem above, followed by an epigraph transformation, we have

$$\begin{aligned}\max_{\substack{z\in\mathbb{Z}(\hat{w})\\ h\in\mathbb{H}}}\min_{\alpha,\beta,\lambda}\quad & \beta\\ \text{s.t.}\quad & \langle\theta,\phi(\hat{s},\hat{x})\rangle + \alpha + \langle\lambda, \hat{c}-\hat{B}z\rangle - \langle q,\phi_2(\hat{w},z)\rangle + \langle h,\hat{y}\rangle + d_z(\hat{z},z) \le \beta\\ & \begin{bmatrix} Q_{yy} & Q\phi_1(\hat{w},z)+h+\hat{A}^\top\lambda\\ * & 4\alpha\end{bmatrix} \succcurlyeq 0\\ & \lambda \ge 0.\end{aligned} \tag{2.38}$$

To reformulate (2.38) into a single minimization problem, we use the fact that

$$\left\{\begin{array}{ll}\max\limits_{x\in\{x_1,\ldots,x_M\}}\min\limits_{t,\delta} & t\\ \text{s.t.} & g(\delta,x)\le t\end{array}\right\} = \left\{\begin{array}{ll}\min\limits_{t,\delta_1,\ldots,\delta_M} & t\\ \text{s.t.} & g(\delta_j,x_j)\le t,\quad \forall j\in[M]\end{array}\right\}, \tag{2.39}$$

for a general function g, which follows from the observation that

$$\begin{aligned}\max_{x\in\{x_1,\ldots,x_M\}}\min_{\delta} g(\delta,x) &= \max\{\min_{\delta_1} g(\delta_1,x_1),\ldots,\min_{\delta_M} g(\delta_M,x_M)\}\\ &= \min_{\delta_1,\ldots,\delta_M}\max\{g(\delta_1,x_1),\ldots,g(\delta_M,x_M)\}.\end{aligned}$$

Combining (2.38) with (2.39), we arrive at

$$\begin{aligned}\min\quad & \beta\\ \text{s.t.}\quad & \theta = (\text{vec}(Q_{yy}),\text{vec}(Q),q)\in\Theta,\quad \lambda_{jk}\ge 0,\quad \alpha_{jk},\beta\in\mathbb{R}\\ & \langle\theta,\phi(\hat{s},\hat{x})\rangle + \alpha_{jk} + \langle\lambda_{jk},\hat{c}-\hat{B}z_j\rangle - \langle q,\phi_2(\hat{w},z_j)\rangle + \langle h_k,\hat{y}\rangle + d_z(\hat{z},z_j)\le\beta\\ & \begin{bmatrix} Q_{yy} & Q\phi_1(\hat{w},z_j)+h_k+\hat{A}^\top\lambda_{jk}\\ * & 4\alpha_{jk}\end{bmatrix}\succcurlyeq 0,\end{aligned} \tag{2.40}$$

where the constraints are $\forall(j,k)\in[M]\times[2u]$, $\mathbb{Z}(\hat{w}) := \{z_1,\ldots,z_{M_i}\}$, $M := |\mathbb{Z}(\hat{w})|$, and if $k\le u$ ($k>u$), h_k is the vector of zeros except for the k'th k'th (($k-u$)'th) element, which

is equal to 1 (-1). Finally, plugging (2.40) into problem (2.10), we arrive at

$$\begin{aligned}
&\min \kappa\mathscr{R}(\theta) + \frac{1}{N}\sum_{i=1}^{N}\beta_i \\
&\text{s.t. } \theta = (\mathrm{vec}(Q_{yy}), \mathrm{vec}(Q), q) \in \Theta, \quad \lambda_{ijk} \geq 0, \quad \alpha_{ijk}, \beta_i \in \mathbb{R} \\
&\quad \langle \theta, \phi(\hat{s}^{[i]}, \hat{x}^{[i]})\rangle + \alpha_{ijk} + \langle \lambda_{ijk}, \hat{c}_i - \hat{B}_i z_{ij}\rangle - \langle q, \phi_2(\hat{w}_i, z_{ij})\rangle + \langle h_k, \hat{y}_i\rangle + d_z(\hat{z}_i, z_{ij}) \leq \beta_i \\
&\quad \begin{bmatrix} Q_{yy} & Q\phi_1(\hat{w}_i, z_{ij}) + h_k + \hat{A}_i^\top \lambda_{ijk} \\ * & 4\alpha_{ijk} \end{bmatrix} \succcurlyeq 0,
\end{aligned}$$

where the constraints are $\forall (i,j,k) \in [N] \times [M_i] \times [2u]$.

2.C.5. PROOF OF PROPOSITION 2.16

Let $\theta^\star \in \operatorname{argmin}_{\theta\in\Theta} f(\theta)$. By the definition of an ε-subgradient, we have

$$\begin{aligned}
\mathbb{E}\left[f(\theta_t) - f(\theta^\star)\right] &\leq \mathbb{E}\left[\langle g_{\varepsilon_t}(\theta_t), \theta_t - \theta^\star\rangle + \varepsilon_t\right] \\
&= \mathbb{E}\left[\langle \mathbb{E}\left[\tilde{g}_{\varepsilon_t}(\theta_t) \mid \theta_t\right], \theta_t - \theta^\star\rangle + \varepsilon_t\right] \\
&= \mathbb{E}\left[\langle \tilde{g}_{\varepsilon_t}(\theta_t), \theta_t - \theta^\star\rangle + \varepsilon_t\right],
\end{aligned} \tag{2.41}$$

where the first equality follows from the definition of a stochastic approximate subgradient and the second equality follows from the law of total expectation. Next, we use the standard inequality for the analysis of mirror-descent algorithms [JN11, Proposition 5.1]

$$\langle \tilde{g}_{\varepsilon_t}(\theta_t), \theta_t - \theta^\star\rangle \leq \frac{1}{\eta_t}(\mathscr{B}_\omega(\theta^\star, \theta_t) - \mathscr{B}_\omega(\theta^\star, \theta_{t+1})) + \frac{\eta_t}{2}\|\tilde{g}_{\varepsilon_t}(\theta_t)\|_*^2. \tag{2.42}$$

Summing (2.42) from $t = 1$ to T and taking its expectation, we arrive at

$$\begin{aligned}
\mathbb{E}\left[\sum_{t=1}^{T}\langle \tilde{g}_{\varepsilon_t}(\theta_t), \theta_t - \theta^\star\rangle\right] &\leq \mathbb{E}\left[\sum_{t=1}^{T}\frac{1}{\eta_t}(\mathscr{B}_\omega(\theta^\star, \theta_t) - \mathscr{B}_\omega(\theta^\star, \theta_{t+1})) + \frac{1}{2}\sum_{t=1}^{T}\eta_t\|\tilde{g}_{\varepsilon_t}\|_*^2\right] \\
&\leq \frac{R^2}{\eta_1} + R^2\sum_{t=2}^{T}\left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) + \frac{G^2}{2}\sum_{t=1}^{T}\eta_t \\
&= \frac{R^2}{\eta_T} + \frac{G^2}{2}\sum_{t=1}^{T}\eta_t \leq \left(\frac{R^2}{c} + cG^2\right)\sqrt{T},
\end{aligned} \tag{2.43}$$

where the second line follows from rearranging the sum and the boundedness assumptions, the third line follows from telescoping the sum, and the fourth line follows from the definition of η_t and the inequality $\sum_{t=1}^{T} 1/\sqrt{t} \leq 2\sqrt{T}$. Finally, using Jensen's inequality, we arrive at

$$\begin{aligned}
\mathbb{E}\left[f\left(\frac{1}{T}\sum_{t=1}^{T}\theta_t\right) - f(\theta^\star)\right] &\leq \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T}(f(\theta_t) - f(\theta^\star))\right] \\
&\leq \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T}\langle \tilde{g}_{\varepsilon_t}(\theta_t), \theta_t - \theta^\star\rangle + \sum_{t=1}^{T}\varepsilon_t\right] \\
&\leq \left(\frac{R^2}{c} + cG^2\right)\frac{1}{\sqrt{T}} + \frac{1}{T}\sum_{t=1}^{T}\varepsilon_t,
\end{aligned}$$

where the second inequality follows from (2.41) and the third inequality follows from (2.43). Next, we prove an improved convergence bound when $f = h+\mathscr{R}$, and $\mathscr{R}$ is relative α-strongly convex. The proof is based on [LSB12]. In this case, similar to (2.41), we have

$$\mathbb{E}\left[f(\theta_t) - f(\theta^\star)\right] \leq \mathbb{E}\left[\langle \tilde{g}_{\varepsilon_t}(\theta_t), \theta_t - \theta^\star\rangle + \varepsilon_t - \alpha \mathscr{B}_w(\theta^\star, \theta_t)\right].$$

Invoking (2.42), multiplying both sides by t, and summing the resulting inequality from $t = 1$ to T, we get

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T} t(f(\theta_t) - f(\theta^\star))\right] &\leq \mathbb{E}\left[\sum_{t=1}^{T}\left(\frac{t}{\eta_t}(\mathscr{B}_\omega(\theta^\star,\theta_t) - \mathscr{B}_\omega(\theta^\star,\theta_{t+1})) \right.\right.\\ &\qquad \left.\left. + \frac{t\eta_t}{2}\|\tilde{g}_{\varepsilon_t}\|_*^2 + t\varepsilon_t - \alpha t \mathscr{B}_w(\theta^\star,\theta_t)\right)\right]\\ &\leq \left(\frac{1}{\eta_1} - \alpha\right)\mathscr{B}_\omega(\theta^\star,\theta_1) + \sum_{t=2}^{T}\left(\frac{t}{\eta_t} - \alpha t - \frac{t}{\eta_{t-1}}\right)\mathscr{B}_\omega(\theta^\star,\theta_t)\\ &\quad + \frac{G^2}{2}\sum_{t=1}^{T} t\eta_t + \sum_{t=1}^{T} t\varepsilon_t.\end{aligned}$$

Next, by setting $\eta_t = 2/\alpha(t+1)$, it is easy to show that the sums between parenthesis become nonpositive, and we arrive at

$$\mathbb{E}\left[\sum_{t=1}^{T} t(f(\theta_t) - f(\theta^\star))\right] \leq \frac{G^2}{\alpha}\sum_{t=1}^{T}\frac{t}{(t+1)} + \sum_{t=1}^{T} t\varepsilon_t \leq \frac{G^2 T}{\alpha} + \sum_{t=1}^{T} t\varepsilon_t. \tag{2.44}$$

Finally, using Jensen's inequality, we have that

$$\begin{aligned}\mathbb{E}\left[f\left(\frac{2}{T(T+1)}\sum_{t=1}^{T} t\theta_t\right) - f(\theta^\star)\right] &\leq \frac{2}{T(T+1)}\mathbb{E}\left[\sum_{t=1}^{T} t(f(\theta_t) - f(\theta^\star))\right]\\ &\leq \frac{2G^2}{\alpha(T+1)} + \frac{2}{T(T+1)}\sum_{t=1}^{T} t\varepsilon_t,\end{aligned}$$

where the second inequality follows from (2.44).

2.C.6. PROOF OF LEMMA 2.17

This proof is based on [Ber15, Example 3.3.1]. Recall the definition of ℓ_θ:

$$\ell_\theta(\hat{s},\hat{x}) = \max_{x\in\mathbb{X}(\hat{s})}\left\{\langle\theta, \phi(\hat{s},\hat{x}) - \phi(\hat{s},x)\rangle + d(\hat{x},x)\right\}.$$

Then, by the definition of x_ε we have

$$\begin{aligned}\ell_\theta(\hat{s},\hat{x}) - \ell_\nu(\hat{s},\hat{x}) &= \max_{x\in\mathbb{X}(\hat{s})}\left\{\langle\theta, \phi(\hat{s},\hat{x}) - \phi(\hat{s},x)\rangle + d(\hat{x},x)\right\}\\ &\qquad - \max_{x\in\mathbb{X}(\hat{s})}\left\{\langle\nu, \phi(\hat{s},\hat{x}) - \phi(\hat{s},x)\rangle + d(\hat{x},x)\right\}\\ &\leq \langle\theta, \phi(\hat{s},\hat{x}) - \phi(\hat{s},x_\varepsilon)\rangle + d(\hat{x},x_\varepsilon) + \varepsilon\\ &\qquad - \max_{x\in\mathbb{X}(\hat{s})}\left\{\langle\nu, \phi(\hat{s},\hat{x}) - \phi(\hat{s},x)\rangle + d(\hat{x},x)\right\}.\end{aligned}$$

Next, by the optimality of the max, we arrive at

$$\begin{aligned}\ell_\theta(\hat{s},\hat{x}) - \ell_\nu(\hat{s},\hat{x}) &\le \langle \theta, \phi(\hat{s},\hat{x}) - \phi(\hat{s},x_\varepsilon)\rangle + d(\hat{x},x_\varepsilon) + \varepsilon - \langle \nu, \phi(\hat{s},\hat{x}) - \phi(\hat{s},x_\varepsilon)\rangle + d(\hat{x},x_\varepsilon) \\ &= \langle \phi(\hat{s},\hat{x}) - \phi(\hat{s},x_\varepsilon), \theta - \nu\rangle + \varepsilon.\end{aligned}$$

This shows that $\phi(\hat{s},\hat{x}) - \phi(\hat{s},x_\varepsilon)$ is an ε-subgradient of ℓ_θ w.r.t. θ.

2.D. FURTHER NUMERICAL RESULTS

2.D.1. BREAST CANCER WISCONSIN PROGNOSTIC DATASET

In this numerical experiment, we present a way to model a real-world decision problem using our IO approaches. Namely, we use the Breast Cancer Wisconsin Prognostic (BCWP) dataset from the UCI Machine Learning Repository [DG17]. This dataset consists of real-world data from different breast cancer patients. For each patient, there are 30 features extracted from images of cells taken from breast lumps, plus the tumor size and the number of involved lymph nodes, for a total of 32 numerical features. Moreover, each case is classified as "recurrent" if the disease has recurred after surgery, and "non-recurrent" if the disease did not recur by the time of the patient's last check-up. For recurrent patients, there is also information on the time to recur (TTR), that is, how many months it took for the disease to recur. For non-recurrent patients, we have information on the disease-free survival time (DFST), that is, how many months it is known that the patient was disease-free after the surgery. The dataset consists of 198 distinct cases, 47 of which have recurred. Thus, given the 32 features, the goal is to predict if the disease will recur or not, as well as the TTR (if recurrent) or the DFST (if non-recurrent).

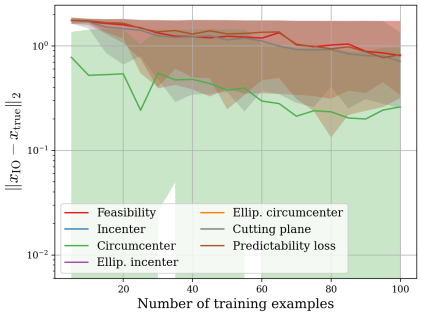
We interpret this problem as an IO problem: given a signal vector $\hat{w} \in \mathbb{R}^{32}$ (numerical vector with the 32 features), an expert agent (e.g., a doctor) returns a decision $(y,z) \in \mathbb{R}\times\{0,1\}$, where y is the TTR/DFST and $z=1$ if the expert predicts the disease to return (recurrent patient), and $z=0$ otherwise (non-recurrent patient). In other words, given the data on some patients, the expert predicts if the disease will recur or not. If recurrent, the expert also predicts the TTR; if not recurrent, the expert predicts the DFST, which can be interpreted as a lower bound on how long the patient is expected to be disease-free. Thus, given the dataset of signal-decision data, we use IO to learn a cost function that when minimized, mimics the behavior of the expert. Since the decision of the expert comprises of continuous and discrete components (i.e., $\mathbb{X}(\hat{s}) = \{(y,z)\in\mathbb{R}\times\{0,1\} : y \ge 0\}$), we tackle this scenario using the mixed-integer approach proposed in Section 2.2.2. Here we note that since the BCWP dataset comes from a real-world scenario, the choice of hypothesis function we use for the IO approach (in particular, the feature functions ϕ_1 and ϕ_2) is a modeling choice. We use the hypothesis function (2.14), with $\phi_1(w,z) = \phi_2(w,z) = (w,z,zw,1)$. Thus, we can use the reformulation (2.15) to solve the problem, with $\hat{A} = -I$, $\hat{B} = \hat{c} = 0$, $\Theta = \mathbb{R}^n$, $d_z(\hat{z}_i,z_i) = |\hat{z}_i - z_i|$, and $\mathscr{R}(\theta) = \frac{1}{2}\|\theta\|_2^2$. We call this approach **ASL-yz**. Similar to the experiments in Section 2.2.2, we also test an approach similar to **ASL-yz**, but with $h_k = 0\ \forall k \in [2u]$ (see discussion in the paragraph after Corollary 2.13). We call this approach **ASL-z**. Finally, as a benchmark, we test a regression+classification approach, where we treat the problem as a separate regression (prediction of the TTR/DFST) and classification (recurrent/non-recurrent patient). For the regression task, we tested scikit-learn's epsilon-support vector regression,

Approach	ASL-z	ASL-yz	regression+classification
$\lvert y - y_{\textbf{true}} \rvert$	51.17	27.33	27.44
$\lvert z - z_{\textbf{true}} \rvert$	20%	21%	21.25%

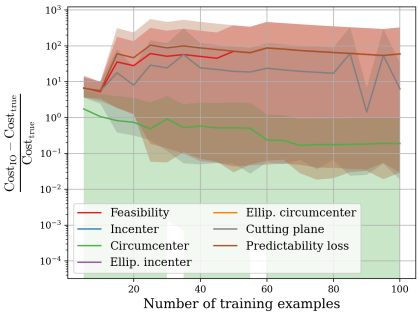
Table 2.4: Out-of-sample average error in months of the TTR/DFST prediction and out-of-sample average percentage error in the prognostic of recurrent vs non-recurrent patients.

ordinary least squares linear regression, kernel ridge regression, multi-layer perceptron regressor, regression based on k-nearest neighbors, gaussian process regression, and a decision tree regressor. For the classification task, we tested scikit-learn's support vector classifier, logistic regression classifier, classifier implementing the k-nearest neighbors vote, gaussian process classification based on Laplace approximation, decision tree classifier, random forest classifier, multi-layer perceptron classifier, and an AdaBoost classifier. For this experiment, using scikit-learn's default tuning parameters, the methods with the best out-of-sample performance for the regression and classification tasks were the kernel ridge regression and support vector classifier, respectively.

To evaluate our approach, we generate 20 training and test datasets, where each training dataset is generated by randomly sampling 90% of the original dataset, while the remaining 10% is used as the test dataset, similar to a 20-fold cross-validation procedure. Table 2.4 shows the out-of-sample results for approaches to this problem. In particular, it shows the average error for the continuous part of the decision variable (he average error in months of the TTR/DFST prediction), i.e., $(1/N)\sum_{i=1}^{N} |y_i - \hat{y}_i|$, where N is the size of the dataset, y_i is the predicted value, and $\hat{y}_i$ is the true value from the dataset, and the percentage error of the discrete part of the decision variable (the average error in the prognostic of recurrent vs non-recurrent patients), i.e., $(100/N)\sum_{i=1}^{N} |z_i - \hat{z}_i|$, where N is the size of the dataset, z_i is the predicted value, and $\hat{z}_i$ is the true value from the dataset. From these results, we can see that the ASL-yz approach has slightly better performance than the regression+classification approach and has a much smaller regression error than the ASL-y approach. Compared to other works that used the BCWP dataset to predict the cancer recurrence or recurrence time, our IO approach differs from them in two major ways: first, our approach requires no pre-processing of the data. For instance, [ZMA06] splits the data into different classes according to the TTR/DFST and learns one predictor per class, and [SMW95] pre-processes the dataset using a feature selection procedure. In contrast, our OI approach requires no pre-processing of the dataset, although we believe its results may be improved after an appropriate pre-processing of the dataset. Second, our IO approach can predict the TTR/DFST and recurrence/non-recurrence *simultaneously*. In machine learning terms, our IO approach does classification and regression at the same time. This differs from the approaches in the literature, which focus on either predicting the TTR/DFST or predicting recurrence/non-recurrence. For instance, [SMW95] was able to achieve an average decision error of 13.9 months using leave-one-out testing and a feature selection procedure, and [LM01] was able to achieve a 16.53% prediction error for positive vs negative cases, where a case is considered positive if a recurrence occurred before 24 months, and negative otherwise.



(a) Average error between the decision generated by θ_{true} and θ_{IO}.



(b) Relative difference between the cost of the decisions generated using θ_{true} and θ_{IO}.

Figure 2.8: In-sample results for consistent data scenario. The results for the incenter approach do not appear in the plots because they are zero for all number of training examples tested.

2.D.2. IN-SAMPLE RESULTS

In this section, we show the in-sample results for the numerical experiments of Section 2.4. Namely, Figure 2.8 shows the in-sample counterpart of Figure 2.3, Figure 2.9 shows the in-sample counterpart of Figure 2.4, Figure 2.10 shows the in-sample counterpart of Figure 2.5, Figure 2.11 shows the in-sample counterpart of Figure 2.12, and Table 2.5 shows the in-sample counterpart of Table 2.4. Notice that since we always learn the IO cost vector using the training data, the difference between the true cost vector and the one learned using IO is independent of in- or out-of-sample results, thus, this plot is not shown in this section. Moreover, for the cases when the training dataset is noisy, we do not show the plot that compares the difference between the cost of the expert decisions with the cost of the decisions using θ_{IO}.

Approach	ASL-z	ASL-yz	regression+classification
$\lvert y - y_{\textbf{true}}\rvert$	34.27	26.06	25.03
$\lvert z - z_{\textbf{true}}\rvert$	24.16%	24.16%	23.48%

Table 2.5: In-sample average error in months of the TTR/DFST prediction and out-of-sample average percentage error in the prognostic of recurrent vs non-recurrent patients.

2.D.3. IO RESULTS FOR SAMD

Figure 2.12 shows the performance of the algorithms tested in Section 2.3.3 in terms of the IO performance metrics. The discussion on the interpretation of the results of Figure 2.12 mirrors the one related to Figure 2.3 from the previous section, with the difference that now the x-axis of the figures refers to running time instead of the number of training examples. In these plots, we can see that the improvements in convergence speed of mirror descent updates, stochastic subgradients, and approximate subgradients are also reflected in the performance of the resulting solution for the IO problem.

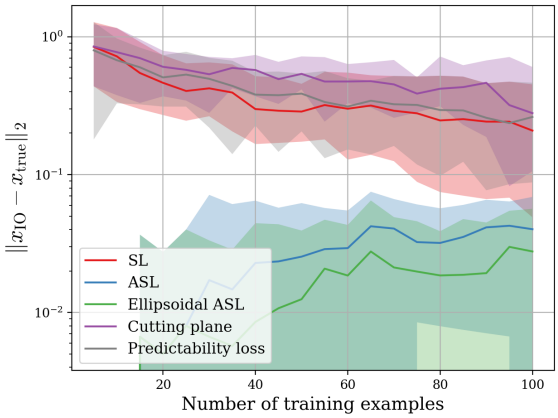
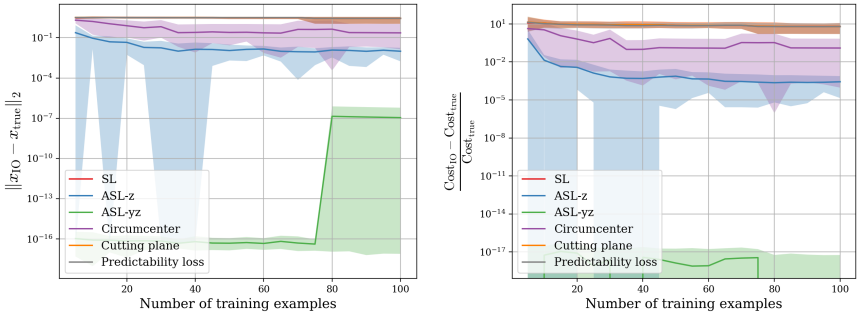


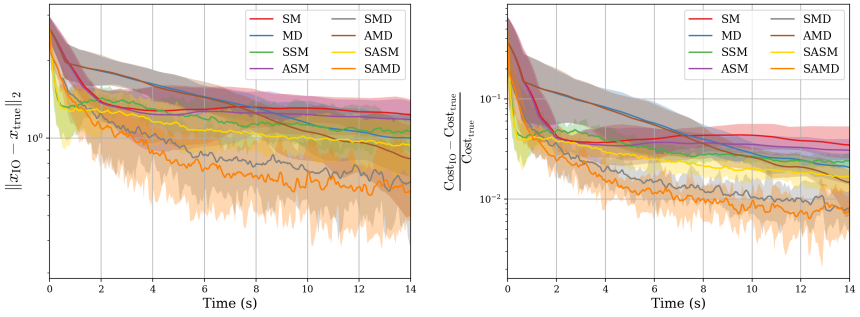
Figure 2.9: In-sample results for inconsistent data scenario. Average error between the decision generated by θ_{true} and θ_{IO}.



(a) Average error between the decision generated by θ_{true} and θ_{IO}.

(b) Relative difference between the cost of the decisions generated using θ_{true} and θ_{IO}.

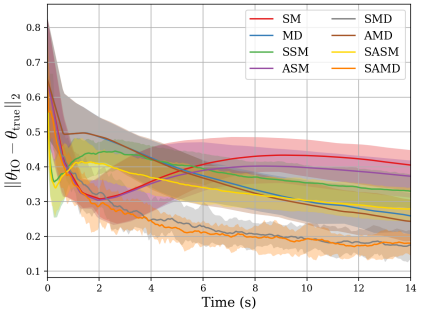
Figure 2.10: In-sample results for the mixed-integer feasible set scenario.



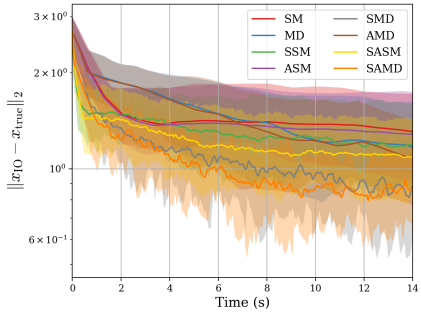
(a) Average error between the decision generated by θ_{true} and θ_{IO}.

(b) Relative difference between the cost of the decisions generated using θ_{true} and θ_{IO}.

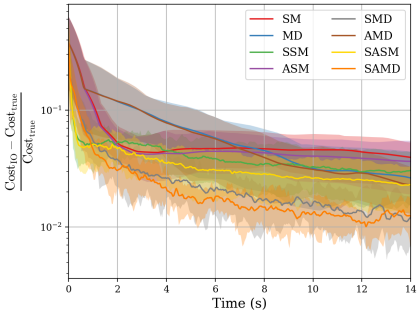
Figure 2.11: In-sample results using iterative first-order algorithms.



(a) Difference between the true cost vector (θ_{true}) and the one learned using IO (θ_{IO}).



(b) Average error between the decision generated by θ_{true} and θ_{IO}.



(c) Relative difference between the cost of the decisions generated using θ_{true} and θ_{IO}.

Figure 2.12: Out-of-sample results using first-order algorithms.

3

Inverse Optimization for Routing Problems

Last-mile delivery is the last stage of delivery in which shipments are brought to end customers. Optimizing delivery routes is a well-researched topic, but most of the classical approaches for this problem focus on minimizing the total travel time, distance, and/or cost of the routes. However, the routes driven by expert drivers often differ from the routes that minimize a time or distance criterion. This phenomenon is related to the fact that human drivers take many different factors into consideration when choosing routes, e.g., good parking spots, support facilities, gas stations, avoiding narrow streets or streets with slow traffic, etc. This contextual knowledge of expert drivers is hard to model and incorporate into traditional optimization strategies, leading to expert drivers choosing potentially more convenient routes under real-life operational conditions, contradicting the optimized route plans. Thus, developing models that capture and effectively exploit this tactic knowledge could significantly improve the real-world performance of optimization-based routing tools. For instance, in 2021, Amazon.com, Inc. proposed the *Amazon Last Mile Routing Research Challenge* [Ama21a] (referred to as the Amazon Challenge in the following). For this challenge, Amazon released a dataset of real-world delivery requests and the respective human routes. The goal was for participants to propose novel methods that use this historical data to learn how to route like an expert human driver, thus incorporating their experience and knowledge when routing vehicles for new delivery requests.

In the literature, several approaches have been proposed to incorporate information from historical route data into the planning of new routes. Some of those methods use discrete choice models and the routes of the drivers are used to determine a transition probability matrix [FFK13]. For instance, in [CG19; Can+21; CMB21; Can+24], a Markov chain framework is used to learn the weights associated with each edge of the graph, which are interpreted as the likelihood of that arc appearing in the optimal solution of

This chapter is based on [Zat+24].

the routing problem. Other approaches use inverse reinforcement learning to learn a routing policy that approximates the ones from historical data [Wul+17; Liu+20]. The Technical Proceedings of the Amazon Challenge [WPN21] contains 31 articles with approaches that were submitted to the Amazon Challenge. Many of these approaches rely on learning specific patterns in the sequence of predefined geographical city zones visited by expert drivers. The paper [Wu+22] uses a sequential probability model to encode the drivers' behavior and uses a policy iteration method to sample zone sequences from the learned probability model. The paper [Pit21] develops an Inverse Reinforcement Learning (IRL) approach for the Amazon Challenge, which despite its name, does not share many similarities with our Inverse Optimization (IO) approach. In particular, in this approach, the TSP is interpreted as a Dynamic Programming (DP) problem, thus, the goal of IRL is to learn the stage cost of this DP from example TPS routes. However, DPs are known to suffer from the curse of dimensionality, that is, these problems become intractable to solve when the dimension of the problem becomes too large (such as for the TSP from the challenge). These issues are reflected in the poor performance of the submissions that use IRL. The IRL method in [SSC21] is closer to our IO methodology, in the sense that a weight matrix is learned from data. However, different from our IO approach, which learns the entire weight matrix simultaneously, they use a Neural Network to map node features to a single edge weight, thus, not accounting for the features of neighboring edges. A successful approach to tackle the challenge was to adjust the travel time matrix between zones based on patterns observed in the training dataset. In particular, both the second-place [GMW21] and third-place [AA21] submissions used this approach. Namely, they extracted rules (i.e., patterns observed in the behavior of the human drivers) through descriptive analysis of the training dataset, and based on these rules, they derived "discouragement multipliers", which are simply constants that multiply each value of the travel time matrix. These multipliers were tuned so that the TSP routes computed using the modified travel times enforce the rules previously extracted.

The IO approach presented in this chapter shares similarities with [GMW21] and [AA21], in the sense that it also uses penalization constants to enforce the behaviors observed in the data. However, differently from them, we combine these penalizations with a custom weight matrix *learned* using IO. The IO methodology in this chapter can be interpreted as a way to combine information extracted from a descriptive analysis of the data with information automatically learned from the data. The authors in [CL21] mention IO as a potential method to effectively tackle the challenge, however, due to the complexity of developing a tailored IO methodology for routing problems, the authors instead used standard ML techniques. The approach that won the Amazon Challenge is based on a constrained local search method, where given a new delivery request, they extract precedence and clustering constraints by analyzing similar historical human routes in the training dataset [CHH22]. Thus, their model is *nonparametric*, in the sense that the entire training dataset is required whenever the route for a new delivery request needs to be computed. This is in contrast with our *parametric* IO model, that is, our model is parametrized by a learned vector of parameters, with a dimension that does not depend on the number of examples in the training dataset.

3.1. TAILORED INVERSE OPTIMIZATION METHODOLOGY

In this section, we define our IO methodology for routing problems in terms of a tailored hypothesis class, loss function, and first-order algorithm.

3.1.1. AFFINE HYPOTHESIS CLASS

Since we can only search for cost functions in a restricted function space and given our focus on routing problems, , in this chapter we consider cost functions in an *affine hypothesis space with a nonnegative cost vector*

$$\mathscr{H}_\theta := \{\langle \theta, x\rangle + h(\hat{s}, x) : \theta \geq 0\}, \tag{3.1}$$

where $\theta \in \mathbb{R}^p$ is the cost vector that parametrizes the cost function, and the affine term $h : \mathbb{S} \times \mathbb{X} \to \mathbb{R}$ is a function that can be used to model terms in the hypothesis function $\langle \theta, x\rangle + h(\hat{s}, x)$ that do not depend on θ. This affine function class generalizes the standard linear hypotheses common in the literature of IO and is a key component of our IO methodology to achieve state-of-the-art results in real-world problems (Section 3.4.1). Moreover, we consider nonnegative cost vectors because, for routing problems, they represent the weights of the edges of a graph. For instance, given a complete graph with n nodes, common cost functions to routing problems are the *two-index* or *three-index* formulations

$$\langle \theta, x\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n} \theta_{ij} x_{ij} \quad \text{and} \quad \langle \theta, x\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{K} \theta_{ijk} x_{ijk},$$

where x_{ij} and x_{ijk} are binary variables equal to 1 if the edge connecting node i to node j is used in the route, and 0 otherwise (for the three-index formulation, we have an extra index k specifying which of the K available vehicles uses the edge) [TV02]. Moreover, we could also have an affine term, for instance, $h(\hat{s}, x) = \sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}(\hat{s}) x_{ij}$ in the cost function, where the term $M_{ij}(\hat{s}) \in \mathbb{R}$ can be used to encode some behavior we would like to enforce in the model. In summary, our goal is to learn a cost vector θ such that when solving the *Forward Optimization Problem* (FOP)

$$\text{FOP}(\theta, \hat{s}) := \arg\min_{x \in \mathbb{X}(\hat{s})} \big\{\langle \theta, x\rangle + h(\hat{s}, x)\big\}, \tag{3.2}$$

we can reproduce (or approximate) the response the expert would have taken when solving the unknown optimization problem (1.1), given the same signal $\hat{s}$.

3.1.2. TAILORED LOSS FUNCTION

Given a signal-response dataset $\{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^{N}$, in this chapter we propose to solve the IO problem (i.e., find a parameter vector θ) by solving a loss minimization problem:

$$\min_{\theta \geq 0} \frac{1}{N}\sum_{i=1}^{N} \ell_\theta(\hat{s}^{[i]}, \hat{x}^{[i]}), \tag{3.3}$$

where $\ell_\theta : \mathbb{S} \times \mathbb{X} \to \mathbb{R}$ is the *loss function*. Using the affine hypothesis class (3.1), we propose the following loss function

$$\ell_\theta(\hat{s}, \hat{x}) := \langle \theta, \hat{x}\rangle + h(\hat{s}, \hat{x}) - \min_{x \in \mathbb{X}(\hat{s})} \big\{\langle \theta + 2\hat{x} - \mathbb{1}, x\rangle + h(\hat{s}, x) - \langle \mathbb{1}, \hat{x}\rangle\big\}, \tag{3.4}$$

where $\mathbb{1} \in \mathbb{R}^p$ is the all-ones vector. The loss function (3.4) is related to Augmented Suboptimality Loss (ASL), differing from the ASL in two ways: (i) it uses the affine hypothesis class introduced in Section 3.1.1, which allows us to effectively use it for a wider range of practical problems (e.g., the Amazon Challenge), and (ii) its inner minimization problem has a convex objective function w.r.t. to x (assuming h is convex in x), in contrast to the case for the ASL, which is nonconvex general. Having an inner minimization problem with convex cost makes its use much more practical since the inner optimization problem has to be solved to evaluate or compute gradients of (3.4). For example, when using first-order methods to optimize it, the inner minimization problem must be solved at each iteration of the algorithm (e.g., see Algorithm 2). This "nonconvex to convex" reformulation is possible by exploiting the fact that routing problems can be modeled using binary decision variables (e.g., $x_{ij} = 1$ if the edge connecting nodes i and j is used, and $x_{ij} = 0$ otherwise). This reformulation is formalized in Proposition 3.1.

Proposition 3.1 (Connection between the ASL and (3.4))**.** *Assume $\mathbb{X} \subseteq \{0,1\}^p$, that is, the decision variables of the FOP are binary. Then, the loss function (3.4) is equivalent to the ASL, if the linear hypothesis $\langle \theta, \phi(\hat{s}, \hat{x}) \rangle$ is substituted by the affine hypothesis $\langle \theta, \hat{x} \rangle + h(\hat{s}, \hat{x})$, and the distance function $d(\hat{x}, x) = \|\hat{x} - x\|_1$.*

Proof. The ASL with the linear hypothesis substituted by the affine hypothesis and $d(\hat{x}, x) = \|\hat{x} - x\|_1$ is equal to $\langle \theta, \hat{x} \rangle + h(\hat{s}, \hat{x}) - \min_{x \in \mathbb{X}(\hat{s})} \{\langle \theta, x \rangle + h(\hat{s}, x) - \|\hat{x} - x\|_1\}$. Next, notice that for binary variables $a, b \in \{0,1\}$, we have the identity $|a - b| = (1 - a)b + (1 - b)a$. Thus, for two binary vectors $\hat{x}, x \in \{0,1\}^p$, using the definition of the ℓ_1-norm, we have that $\|\hat{x} - x\|_1 = \langle \mathbb{1} - 2\hat{x}, x \rangle + \langle \mathbb{1}, \hat{x} \rangle$. □

3.1.3. FIRST-ORDER ALGORITHM

In this chapter, we propose to solve problem (3.3) using a stochastic first-order algorithm. In particular, our algorithm uses update steps tailored to the proposed loss function (3.4) with a nonnegative cost vector. Moreover, it exploits the finite sum structure of the problem (i.e., the sum over the N examples) by using a single training example per iteration of the algorithm. To do so, we propose a reshuffled sampling strategy, which empirically outperforms the uniform sampling strategy of the SAMD algorithm (see results in Sections 3.2.2 and 3.4.1).

Before presenting our algorithm, we discuss how to compute subgradients of the loss function (3.4), which is necessary to use first-order methods to minimize it. To this end, we define

$$\text{A-FOP}(\theta, \hat{s}, \hat{x}) := \arg\min_{x \in \mathbb{X}(\hat{s})} \{\langle \theta + 2\hat{x} - \mathbb{1}, x \rangle + h(\hat{s}, x)\}, \tag{3.5}$$

that is, the set of optimizers of the FOP with *augmented* edge weights $\theta + 2\hat{x} - \mathbb{1}$ instead of θ. Being able to solve the augmented FOP (3.5) is important because to compute a subgradient of the loss (3.4) (and thus, a subgradient of (3.3)), we need to compute an element of A-FOP$(\theta, \hat{s}, \hat{x})$, which follows from Danskin's theorem [Ber08, Section B.5]. Here we emphasize an important consequence of the reformulation that led to our tailored loss function: assuming the affine term is linear in x, say $h(\hat{s}, x) = \langle M(\hat{s}), x \rangle$, solving the A-FOP has the same complexity as solving the FOP. For example, if the FOP is a TSP with edge weights θ, then the A-FOP is also a TSP, but with augmented edge weights

$\theta + 2\hat{x} - \mathbb{1} + M(\hat{s})$. This is of particular practical interest since the same solver can be used both for *learning* the model (i.e., for the A-FOP, which we must be able to solve to evaluate (3.4), thus, to solve (3.3)) and for *using* the model (i.e., for the FOP).

Algorithm 2 Reshuffled stochastic first-order algorithm

1: **Input:** Step-size sequence $\{\eta_t\}_{t=1}^{T}$, initial point $\theta_1^{[1]} \geq 0$, number of epochs T, and dataset $\{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^{N}$
2: **for** $t = 1, \ldots, T$ **do**
3: Sample $\{\pi_1, \ldots, \pi_N\}$, a permutation of $\{1, \ldots, N\}$
4: **for** $i = 1, \ldots, N$ **do**
5: $x^\star \in \text{A-FOP}\left(\theta_t^{[i]}, \hat{s}^{[\pi_i]}, \hat{x}^{[\pi_i]}\right)$
6: $g = \hat{x}^{[\pi_i]} - x^\star$
7:

$$\theta_t^{[i+1]} = \begin{cases} \theta_t^{[i]} \odot \exp(-\eta_t g) & \text{(exponentiated update)} \\ \text{OR} & \\ \max\left\{0,\ \theta_t^{[i]} - \eta_t g\right\} & \text{(standard update)} \end{cases}$$

8: **end for**
9: $\theta_{t+1}^{[1]} = \theta_t^{[N+1]}$
10: **end for**
11: **Output:** $\left\{\theta_t^{[N+1]}\right\}_{t=1}^{T}$

Algorithm 2 shows our *reshuffled stochastic first-order algorithm* to solve Problem (3.3) using the loss function (3.4). The algorithm runs for T epochs. The number of epochs T should be viewed as an input parameter of Algorithm 2. In practice, one can choose T to be as large as possible and then monitor the performance of the learned model after each epoch of the algorithm. This way, the algorithm is evaluated for all epochs up until T, and we can choose the model with the best performance. Alternatively, we can use Algorithm 2 without a predefined number of epochs T and run it until a stopping criterion is reached. In practice, this could be implemented by running the algorithm until the difference in the test dataset performance (or any other performance metric) of the models from epochs t and $t+1$ is smaller than a minimum value. For instance, in Figure 3.10a, we can see that between epochs $T = 4$ and $T = 5$, the Amazon score of the learned models does not change much, thus, we could use it to stop the algorithm. In our numerical experiments, we chose a predefined T and monitored the performance of the model after each epoch. At the beginning of each epoch, we sample a permutation of $[N]$ (line 3), which simply means that we shuffle the order of the examples in the dataset. This is known as *random reshuffling*, as has been shown to perform better in practice compared to standard uniform stochastic sampling [MKR20]. Moreover, since our random reshuffling strategy uses only one example per update step, it is

particularly efficient for problems with large datasets. Next, for each epoch, we perform one update step for each example in the dataset. In particular, in line 5 we compute one element of A-FOP, and in line 6 we compute a subgradient of the loss function (3.4). For the update step (line 7), we offer two possibilities: (i) *exponentiated updates*, which are inspired by the exponentiated subgradient algorithm of [KW97] and are specialized for optimization problems with nonnegative variables, and (ii) *standard updates*, which can be interpreted as using the standard projected subgradient method, projecting onto the nonnegative cone. In practice, the question of what update step is the best should be answered on a case-by-case basis. An important component of our algorithm (and of first-order algorithms in general) is the step size η_t. Common choices are $\eta_t = c/\sqrt{t}$, $\eta_t = c/t$, or $\eta_t = c$, for some fixed constant $c > 0$. Finally, to turn the output of the algorithm $\{\theta_t^{[N+1]}\}_{t=1}^T$ into a single cost vector, we can use standard methods such as $\theta = \theta_T^{[N+1]}$ (last iterate), $\theta = \frac{1}{T}\sum_{t=1}^T \theta_t^{[N+1]}$ (average) or $\theta = \frac{2}{T(T+1)}\sum_{t=1}^T t\theta_t^{[N+1]}$ (weighted average) [LSB12].

Remark 3.2 (Approximate A-FOP)**.** *Notice that an element of A-FOP needs to be computed at each iteration of Algorithm 2 (line 5). However, if the A-FOP is a hard combinatorial problem (e.g., a large TSP or VRPTW), it may not be computationally feasible to solve it to optimality multiple times. Thus, in practice, one may use an* approximate A-FOP*, that is, in line 5 of Algorithm 2 we compute an approximate solution to the augmented FOP instead of an optimal one. Fortunately, an approximate solution of the A-FOP can be used to construct an* approximate subgradient*, which in turn can be used to compute an approximate solution of Problem* (3.3) *(see Section 2.3.3). In practice, using approximate solvers may lead to a much faster learning algorithm, in exchange for a possibly worse learned model. This trade-off is explored in the numerical results of Section 3.4.2.*

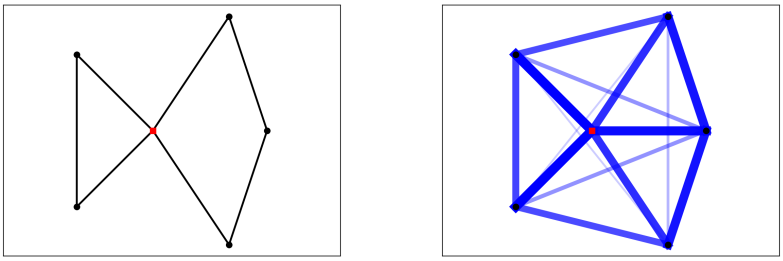
3.2. MODELING EXAMPLES

Next, we present three examples of how our IO methodology can be used for learning from routing data. Namely, we first exemplify how a CVRP scenario can be modeled with our IO methodology, and present a simple numerical example to illustrate the intuition behind how Algorithm 2 works. Second, we show how a larger VRPTW scenario can be modeled with our IO methodology, and present numerical results using data generated from real-world instances. Third, we define a class of TSPs, which will later be used to formalize the Amazon Challenge as an IO problem.

3.2.1. IO FOR CVRPS

We define the K-vehicle Symmetric Capacitated Vehicle Routing Problem (SCVRP) as

$$\begin{aligned} \min_{x_e\in\{0,1\}\ \ \forall e\in E} \quad & \sum_{e\in E} w_e x_e, & \\ \text{s.t.} \quad & \sum_{e\in\delta(i)} x_e = 2 & \forall i \in V\setminus\{0\} \\ & \sum_{e\in\delta(0)} x_e = 2K & \\ & \sum_{e\in\delta(S)} x_e \geq 2r(S,D,c) & \forall S\subset V\setminus\{0\},\ S\neq\emptyset, \end{aligned} \tag{3.6}$$



(a) Optimal SCVRP routes using weights w_e based on Euclidean distances.

(b) Representation of the weights of each edge of the graph, where the smaller the weight, the thicker and darker the edge.

Figure 3.1: Optimal SCVRP tour and representation of graph weights.

where $\mathcal{G} = (V, E, W)$ is an edge-weighted graph, with node set V (node 0 being the depot), undirected edges E, and edge weights W. For this problem, each node $i \in V$ represents a customer with demand $d_i \in D$. There are K vehicles, each with a capacity of c. Given a set $S \subset V$, let $\delta(S)$ denote the set of edges that have only one endpoint in S. Moreover, given a set $S \subset V \setminus \{0\}$, we denote by $r(S, D, c)$ the minimum number of vehicles with capacity c needed to serve the demands of all customers in S. The x_e's are binary variables equal to 1 if the edge $e \in E$ is used in the solution, and equal to 0 otherwise, and $w_e \in W$ is the weight of edge $e \in E$ [TV02].

Next, we show how to use IO to learn edge weights that can be used to replicate the behavior of an expert, given a dataset of example routes. Consider the signal $\hat{s} := D$, where D is a set of demands of the customers, and the response $\hat{x} \in \{0,1\}^{|E|}$, which is the vector with components x_e encoding the optimal solution of the Problem (3.6) for the signal $\hat{s}$. Defining the linear hypothesis function (i.e., $h(\hat{s}, x) = 0$)

$$\langle \theta, x \rangle := \sum_{e \in E} \theta_e x_e, \tag{3.7}$$

and the constraint set

$$\mathbb{X}(\hat{s}) := \left\{ x \in \{0,1\}^{|E|} : \begin{array}{ll} \sum_{e \in \delta(i)} x_e = 2 & \forall i \in V \setminus \{0\}, \\ \sum_{e \in \delta(0)} x_e = 2K, & \\ \sum_{e \in \delta(S)} x_e \geq 2r(S, D, c) & \forall S \subset V \setminus \{0\},\ S \neq \emptyset \end{array} \right\}, \tag{3.8}$$

we can interpret the signal-response pair $(\hat{s}, \hat{x})$ as coming from an expert agent, which given the signal $\hat{s}$ of demands, solves the SCVRP to compute its response $\hat{x}$. Thus, to learn a cost function (i.e., learn a vector of edge weights) that replicates the SCVRP route $\hat{x}$, we can use Algorithm 2 to solve Problem (3.3) with hypothesis (3.7) and constraint set (3.8).

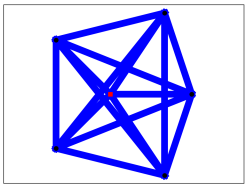
To illustrate how Algorithm 2 works to learn edge weights in routing problems on graphs, consider a simple SCVRP with $K = 2$ vehicles, each with capacity $c = 3$, and 5 customers, each customer i with demand $d_i = 1$. In this example, for simplicity, we use w_e equal to the Euclidean distance between the customers, however, any other set of weights could be used instead. We create one training example using these weights. Figure 3.1a shows the location of the customers (black dots), the depot (red square), and the optimal SCVRP routes using weights w_e. Figure 3.1b shows a representation of the weights of each edge of the graph, where the smaller the weight, the thicker and darker the edge. We use Algorithm 2 with exponentiated updates, $\eta_t = 0.0002$, and we initialize θ_1 with the same weight for all edges. In Figure 3.2, we graphically show two iterations of the algorithm for this problem. In the first column, we show the evolution of the learned weights θ_t. In the second column, we show optimal SCVRP routes computed using the weights in the first column (i.e., computed by solving the A-FOP in line 5 of Algorithm 2), and in the third column, we show the difference between the optimal routes using the true weights (Figure 3.1a) and the optimal routes using the current learned weights (the route in the second column). The difference between these two routes is the subgradient computed in line 6 of Algorithm 2, which is used to update the learned weights θ_t. For the subgradient representation in the third column, red edges represent a negative subgradient (i.e., edges with weights that should be increased) and green edges represent a positive subgradient (i.e., edges with weights that should be decreased). This is the main intuition behind Algorithm 2: at each iteration, we compare the route we want to replicate with the one we get with the current edge weights. Then, comparing which edges are used in these two routes, we either increase or decrease their respective weights, thus "pushing" the optimal route using the learned weights to be closer to the route we want to replicate. In the example shown in Figure 3.2, we can see that after two iterations of the Algorithm 2, the optimal route using the learned weights coincides with the example route.

3.2.2. IO FOR VRPTWS

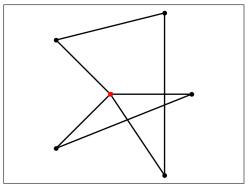
Consider the Vehicle Routing Problem with Time Windows (VRPTW)

$$\begin{aligned} \min_{x_{ijk}\in\{0,1\} \quad \forall i,j,k\in[n]\times[n]\times[K]} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{K} w_{ij}x_{ijk} \\ \text{s.t.} \quad & x \in \mathbb{X}(\hat{s}), \end{aligned} \tag{3.9}$$

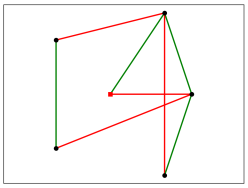
where n is the number of customers, K is the maximum number of vehicles available, x_{ijk} is a binary variable equal to 1 if the edge from node i to node j is traversed by vehicle k in the solution, and 0 otherwise, and w_{ijk} is the weight of the edge connecting node i to node j. In the constraint set of program (3.9), x is the vector containing the variables x_{ijk}, the signal $\hat{s}$ is defined to be the list of time windows (one for each customer) that need to be respected, and $\mathbb{X}(\hat{s})$ is the set of feasible solutions for the VRPTW for time windows in $\hat{s}$. Notice that the set $\mathbb{X}(\hat{s})$ may depend on other parameters of the problem, such as the service time of each customer, the demands of each customer, the travel time between customers, etc. However, we make the constraint set explicitly dependent only on the time windows since this is the only external parameter that will change in this



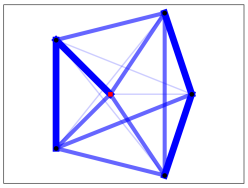
(a) θ_1.



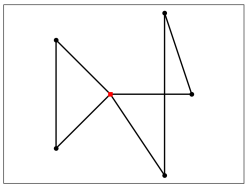
(b) Optimal SCVRP routes using the weights θ_1.



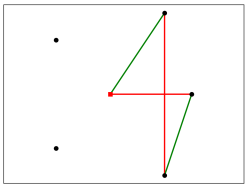
(c) Difference between true optimal route (Figure 3.1) and learned route (Figure 3.2b).



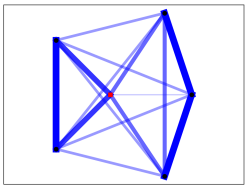
(d) θ_2.



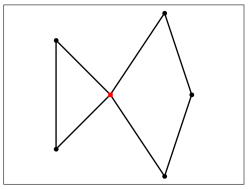
(e) Optimal SCVRP routes using the weights θ_2.



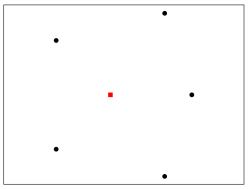
(f) Difference between true optimal route (Figure 3.1) and learned route (Figure 3.2e).



(g) θ_3.



(h) Optimal SCVRP routes using the weights θ_3.



(i) Difference between true optimal route (Figure 3.1) and learned route (Figure 3.2h).

Figure 3.2: Two iterations of Algorithm 2. The figures in the first column represent the learned weights, the figures in the second column are optimal SCVRP routes for the respective weights shown in the first column, and the third column represents the subgradient in line 6 of Algorithm 2, where red (green) edges represent weights that should be increased (decreased).

example. More details on the different formulations for the constraint set of VRPTWs can be found in [TV02].

Next, we show how one can model the VRPTW into our IO framework. Consider the dataset $\{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^{N}$, where the signal $\hat{s}^{[i]}$ is the list of time windows that need to be respected and the response $\hat{x}^{[i]} \in \{0,1\}^{n^2K}$ is the respective optimal VRPTW routes (i.e., a vector with components x_{ijk}). Defining the linear hypothesis function

$$\langle \theta, x \rangle := \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{K} \theta_{ij} x_{ijk}, \tag{3.10}$$

we can interpret this dataset as coming from an expert agent, which given the signal $\hat{s}^{[i]}$, solves a VRPTW to compute its response $\hat{x}^{[i]}$. Thus, to learn a cost function (i.e., learn a vector of edge weights) that replicates the VRPTW route $\hat{x}$, we can use Algorithm 2 to solve Problem (3.3) with hypothesis (3.10) and the constraint set of (3.9).

To illustrate this formulation, we will use a VRPTW scenario generated using data from the *EURO Meets NeurIPS 2022 Vehicle Routing Competition* [ORT22]. The VRPTWs considered in this competition are real-world instances provided by the company ORTEC. To generate the training data to test our IO formulation, we pick one instance from the competition, which corresponds to a relatively large VRPTW with $n = 200$ customers and $K = 15$ available vehicles. Originally, each customer in this VRPTW instance had fixed time windows. However, to generate an IO dataset, we shuffled the original time windows among the 200 customers and computed the optimal VRPTW routes for each of these new instances. Thus, we generate a dataset $\{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^{N}$, where the signal $\hat{s}^{[i]}$ is a random assignment of time-windows to customers, and the response $\hat{x}^{[i]}$ is the respective optimal VRPTW solution. Using the state-of-the-art solver PyVRP [WLK24], we generated $N = 50$ training and test instances. All these instances have the same true edge weights w_{ij}, which corresponds to the non-euclidean real-world road driving time from customer i to customer j. Thus, our IO goal is to learn a set of weights θ_{ij} that replicate the routes using w_{ij} as well as possible, given the provided dataset of signal-response training examples. We learn the weights using the training dataset and evaluate its performance using a test dataset. Moreover, we report the average performance value for 5 randomly generated training/test datasets, as well as the 5th and 95th percentile bounds. We test three approaches to solve the IO problem, where we set the initial weights in θ_1 equal to the Euclidean distance between customers i and j.

- **Cutting plane:** We use the cutting plane algorithm from [Wan09] to solve

$$\begin{aligned} \min_{\theta \geq 0} \quad & \|\theta - \theta_1\|_1 \\ \text{s.t.} \quad & \hat{x}_i \in \text{FOP}(\theta, \hat{s}_i) \quad \forall i \in [N], \end{aligned}$$

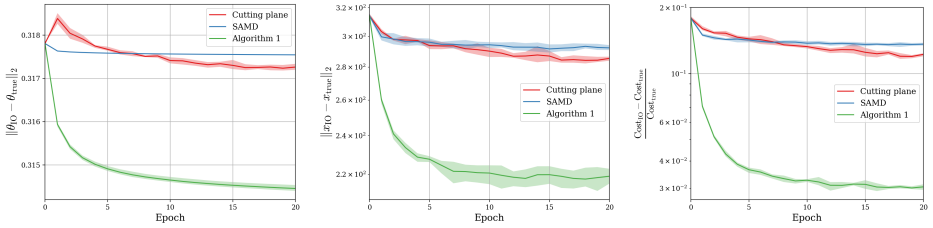
  which is the multi-point IO formulation proposed in [BCZ22].

- **SAMD:** We use the SAMD algorithm to solve (3.3), with exponentiated updates and $\eta_t = 0.3/t$.

- **Algorithm 1:** We use Algorithm 2 to solve (3.3), with exponentiated updates and $\eta_t = 0.3/t$. For this example, the difference between the SAMD algorithm and Algorithm 2 is that the former uses uniform stochastic sampling, while the latter uses the reshuffled sampling strategy.

Our experiments are reproducible, and the underlying source code is available at [Zat23b]. Figure 3.3 shows the results of this experiment. For all the plots, the x-axis refers to the epoch $t \in [1, T]$, which consists of N iterations of the method used to solve the problem. Figure 3.3a shows the normalized difference between the vector of weights returned by the IO approach (which we name θ_{IO}) and the vector of weights used to generate the data (which we name θ_{true}). Figure 3.3b shows the average difference between the optimal routes using θ_{IO} (which we name x_{IO}) and the routes from the test dataset

(which we name x_{true}). Figure 3.3c shows the normalized difference between the cost of the expert decisions and the cost of the decisions using θ_{IO}. More precisely, we define $\text{Cost}_{\text{IO}} := \sum_{i=1}^{N} \langle \theta_{\text{true}}, x_{\text{IO}}^{[i]} \rangle$ and $\text{Cost}_{\text{true}} := \sum_{i=1}^{N} \langle \theta_{\text{true}}, x_{\text{true}}^{[i]} \rangle$ and compare the relative difference between them. Notice that this difference will always be nonnegative by the optimality of $x_{\text{true}}^{[i]}$. From the results of this experiment, we can see that Algorithm 2 outperforms the other approaches (i.e., the cutting plane and SAMD) by a relatively large margin, which shows the efficacy of our proposed reshuffled sampling strategy (i.e., Algorithm 2) for this example.



(a) Difference between the true weights (θ_{true}) and the ones learned using IO (θ_{IO}).

(b) Average error between the routes generated by θ_{true} and θ_{IO}.

(c) Average normalized cost difference between the routes generated by θ_{true} and θ_{IO}.

Figure 3.3: Results for the VRPTW scenario.

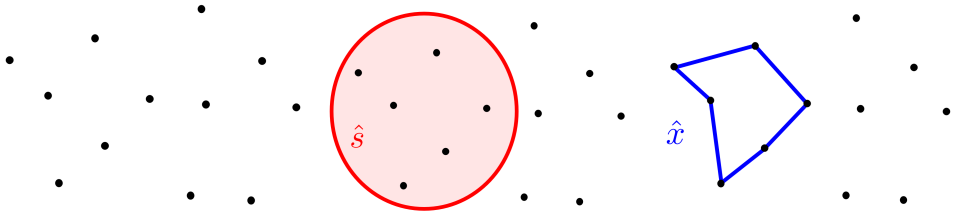
3.2.3. IO FOR TSPS

Let $\mathcal{G} = (V, E, W)$ be a complete edge-weighted directed graph, with node set V, directed edges E, and edge weights W. Next, given $\hat{s} \subset V$ (i.e., a subset of the nodes of $\mathcal{G}$), we define the *Restricted Traveling Salesperson Problem* (R-TSP) as

$$\begin{aligned} \min_{x_{ij}} \quad & \sum_{i \in V} \sum_{j \in V} w_{ij} x_{ij}, \\ \text{s.t.} \quad & \sum_{j \in \hat{s}} x_{ij} = \sum_{j \in \hat{s}} x_{ji} = 1 && \forall i \in \hat{s} \\ & \sum_{i \in Q} \sum_{j \in Q} x_{ij} \leq |Q| - 1 && \forall Q \subset \hat{s},\ Q \neq \emptyset,\ \bar{Q} \neq \emptyset \\ & x_{ij} \in \{0, 1\} && \forall (i, j) \in \hat{s} \times \hat{s} \\ & x_{ij} = 0 && \forall (i, j) \notin \hat{s} \times \hat{s}, \end{aligned} \tag{3.11}$$

where x_{ij} is a binary variable equal to 1 if the edge from node i to node j is used in the solution, and 0 otherwise, and w_{ij} is the weight of the edge connecting node i to node j. Problem (3.11) is based on the standard formulation of a TSP as a binary optimization problem [DFJ54]. The only difference to a standard TSP is that instead of being required to visit all nodes of the graph, for an R-TSP we compute the optimal tour over a subset $\hat{s}$ of the nodes V. Notice that the standard TSP can be interpreted as an R-TSP, for the special case when $\hat{s} = V$. In practice, any TSP solver can be used to solve an R-TSP by simply ignoring all nodes of the graph that are not required to be visited.

Next, we show how to use IO to learn edge weights that can be used to replicate the behavior of an expert, given a set of example routes. Consider the dataset $\{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^{N}$,



(a) Nodes of a graph $\mathscr{G}$. (b) Signal. (c) Expert response.

Figure 3.4: Illustration of signal and expert response for an R-TSP.

where the signal $\hat{s}^{[i]} \in V$ is a set of nodes required to be visited and the response $\hat{x}^{[i]} \in \{0,1\}^{|V|^2}$ is the respective optimal R-TSP tour (i.e., a vector with components x_{ij} for $(i,j) \in V \times V$). Defining the affine hypothesis function

$$\langle \theta, x \rangle + h(\hat{s}, x) := \sum_{i \in V} \sum_{j \in V} \big(\theta_{ij} + M_{ij}\big) x_{ij}, \tag{3.12}$$

and the constraint set

$$\mathbb{X}(\hat{s}) := \left\{ x \in \{0,1\}^{|V|^2} : \begin{array}{ll} \sum_{j \in \hat{s}} x_{ij} = 1, & \forall i \in \hat{s} \\ \sum_{i \in \hat{s}} x_{ij} = 1, & \forall j \in \hat{s} \\ \sum_{i \in Q} \sum_{j \in Q} x_{ij} \leq |Q| - 1, & \forall Q \subset \hat{s}, Q \neq \emptyset, \bar{Q} \neq \emptyset \\ x_{ij} = 0 & \forall (i,j) \notin \hat{s} \times \hat{s} \end{array} \right\}, \tag{3.13}$$

we can interpret this dataset as coming from an expert agent, which given the signal $\hat{s}^{[i]}$, solves an R-TSP to compute its response $\hat{x}^{[i]}$. For the hypothesis function, the term M_{ij} can be used as a penalization term to enforce some kind of expected behavior to the model, e.g., by adding penalizations to some edges of the graph. Figure 3.4 illustrates a signal and expert response for an R-TSP. Thus, to learn a cost function (i.e., learn a vector of edge weights) that replicates (or approximates as well as possible) the example routes in the dataset, we can use Algorithm 2 to solve Problem (3.3) with hypothesis (3.12) and constraint set (3.13). This formulation will serve as the basis of our IO approach to tackle the Amazon Challenge.

We conclude this section with some general comments about our IO approach. First, our IO approach does not require the dataset $\{(\hat{s}^{[i]}, \hat{x}^{[i]})\}_{i=1}^{N}$ to be consistent with a single cost function (i.e. a single set of edge weights), which is to be expected in any realistic setting, due to model uncertainty, noisy measurements or bounded rationality [Moh+18b]. Also, we showed how to use our IO approach for SCVRPs, VRPTWs, and R-TSP scenarios, but we emphasize that the methodology developed in this section could be easily adapted to different kinds of routing problems. For instance, if the problem was a VRP with backhauls, or pickup and delivery locations, we could easily account for these characteristics, for example, by changing the constraint set $\mathbb{X}(\hat{s})$ of our IO model [TV02], or in

other words, by modifying the problem we assumed the expert agent is solving to generate its response. Notice that in any case, the methodology developed in Sections 3.1.1, 3.1.2, and 3.1.3 would not change, which highlights the generality and flexibility of our IO approach. As a final comment, we mention that our approach can easily be adapted to the scenario where new signal-response examples arrive in an *online* fashion. That is, instead of learning from an offline dataset of examples, we gradually update the edge weights (i.e., θ_t) with examples that arrive online, similar to [BPS17]. This can be done straightforwardly by adapting Algorithm 2 to use examples that arrive online in the same way it uses the signal-response pairs $(\hat{s}^{[\pi_i]}, \hat{x}^{[\pi_i]})$ (see Remark 2.18).

3.3. AMAZON CHALLENGE

In this section, we describe the Amazon Challenge, which we use as a real-world application to assess our IO approach. A detailed description of the data provided for the challenge can be found in [Mer+22]. In summary, Amazon released two datasets for this challenge: a training dataset and a test dataset. The training dataset consists of 6112 historical routes driven by experienced drivers. This dataset is composed of routes performed in the metropolitan areas of Seattle, Los Angeles, Austin, Chicago, and Boston, and each route is characterized by several features. Figure 3.5 shows a high-level description of the features available for each example route. Each of these routes starts at a depot, visits a collection of drop-off stops assigned to the driver in advance, and ends at the same depot. Thus, each route can be interpreted as an R-TSP route. Figure 3.6 shows 8 example routes leaving from a depot in Boston, where different colors represent different routes. Each stop in every route was given a Zone ID, which is a unique identifier denoting the geographical planning area into which the stop falls, and is devised internally by Amazon [Mer+22]. Some stops in the dataset are not given a Zone ID, so for these stops, we assign them the Zone ID of the closest zone (in terms of Euclidean distance). Turns out, this predefined zoning of the stops is a key piece of information about the Amazon Challenge. This will be discussed in detail in the subsequent Sections of this chapter.

As previously mentioned, the goal of the challenge was to incorporate the preferences of experienced drivers into the routing of last-mile delivery vehicles. Thus, rather than coming up with TSP strategies that minimize time or distance given a set of stops to be visited, the goal of the challenge was to learn from historical data how to route like the expert drivers. To this end, a test dataset consisting of 3072 routes was also made available to evaluate the proposed approaches. To compare the routes from expert human drivers to the routes generated by the models submitted to the challenge, Amazon devised a scoring metric that computes the similarity between two routes, where the lower the score, the more similar the routes. In particular, if A is the historically realized sequence of deliveries, sequence B is the sequence of deliveries generated by a model, its score is defined as follows:

$$\text{score}(A,B) = \frac{SD(A,B) \cdot ERP_{norm}(A,B)}{ERP_e(A,B)}, \tag{3.14}$$

where sequence SD denotes the Sequence Deviation of B with respect to A, ERP_{norm} denotes the Edit Distance with Real Penalty applied to sequences A and B with normal-

Data field	Description	Unit/format
	Route information	
Route ID	Unique and anonymized identifier of each route	—
Station code	Unique identifier for a delivery station where routes begin	—
Date	Date of route execution	YYYY-MM-DD
Departure time	Time when vehicle leaves the station	—
Executor capacity	Volumetric capacity of vehicle	cm^3
Stops	Each stop on route	—
Observed sequence	Sequence in which stops were visited	—
Route score	Quality of the observed sequence	Categorical
	Stop information	
Stop ID	Unique identifier of each stop on a route	—
Latitude/longitude	Obfuscated coordinates of each stop	—
Type	Type of stop	Categorical
Zone ID	Geographical planning area	—
Packages	Packages delivered at each stop	—
Transit time	Estimated transit time to every other stop on route	Seconds
	Package information	
Package ID	Unique and anonymized identifier of each package	—
Status	Delivery status of package	Categorical
Time window	Start and end of time window, when applicable	—
Planned service time	Time that serving the package is expected to require	Seconds
Dimensions	Length, width, and height of package	cm

Figure 3.5: High-level description of data fields provided in the Amazon Challenge data set [Mer+22].

ized travel times, and ERP_e denotes the number of edits prescribed by the ERP algorithm on sequence B with respect to A. If edit distance with real penalty prescribes 0 edits, then the above formula is replaced by the sequence deviation, multiplied by 0. Thus, the Amazon score combines a similarity measure that takes into account only the sequence of stops in the routes (i.e., SD) with a similarity measure that also takes the travel times between stops into account (i.e., ERP). The details of the score computation can be found at [Ama21b]. Notice that, instead of using our tailored loss function (3.4), one could use the scoring function (3.14) directly to learn the routing model, which makes intuitive sense since minimizing this score is the actual goal of the Amazon Challenge. However, the resulting IO problem would be an intractable bi-level optimization problem, similar to the case when using the so-called *predictablity loss* for IO [ASS18]. This issue highlights one of the big advantages of using our tailored loss function (3.4): the resulting optimization problem is convex and subgradients of the loss function can be computed in closed form, thus, making the problem amendable to be solved using efficient first-order methods, such as Algorithm 2.

In summary, a dataset of 6112 historical routes from expert human drivers was made available for the Amazon Challenge. Using this dataset, the goal is to come up with routing methods that replicate the way human drivers route vehicles. To evaluate the proposed approaches, Amazon used a test dataset consisting of 3072 unseen examples. In order to compare how similar the routes from this dataset are to the ones computed by the submitted approaches, a similarity score was devised. The final score is the average score over the 3072 test instances. A summary of the scores of the top 20 submissions to the Amazon Challenge can be found at [Ama21c]. Since each historical route in the dataset of the challenge refers to a driver's route that starts at a depot, visits a predefined set of customers, and then returns to the depot, the expert human routes from the Amazon Challenge can be interpreted as solutions to R-TSPs, and we can use the IO approach

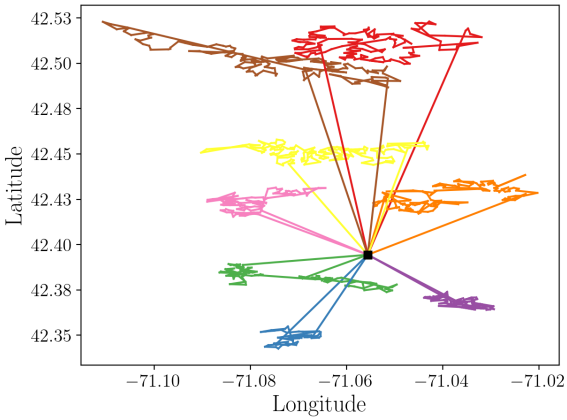


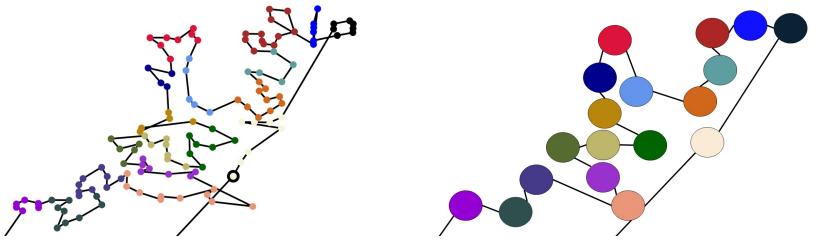
Figure 3.6: Example routes from depot DBO1 in Boston, where different colors represent different routes.

to tackle the Amazon Challenge. In other words, we can use IO to estimate the costs they assign to the street segments connecting stops. Ultimately, this will allow us to learn the drivers' preferences, and replicate their behavior when faced with new requests for stops to be visited.

3.3.1. ZONE IDS AND TIME WINDOWS

In Section 3.2.3, we describe how IO can be used to learn drivers' preferences from R-TSP examples. Although the Amazon Challenge training dataset consists of 6,112 historical routes, it is difficult to learn any meaningful preference of the drivers at the *stop level* (i.e., individual customer level), since the latitude and longitude coordinates of each stop have been anonymized and perturbed to protect the privacy of delivery recipients [Mer+22]. However, recall that each stop in the dataset is assigned a Zone ID, which refers to a geographical zone in the city (see Figure 3.5), and each zone contains multiple stops. Analyzing how the human drivers' routes relate to these zones, a critical observation can be made: in the vast majority of the examples, the drivers visit all stops within a zone before moving to another zone (the zone ID of consecutive stops is the same around 85% of the time). This behavior is illustrated in Figure 3.7a. Also, the same zone is usually visited in multiple route examples in the dataset. Thus, instead of learning drivers' preferences at the stop level, we can learn their preferences at the *zone level*. In other words, we consider each zone as a *hypernode* containing all stops with the same Zone ID. Thus, we can create a *hypergraph* with nodes corresponding to the zone hypernodes (see Figure 3.7). This way, we can view the expert human routes as routes over zones, and we can use our IO approach to learn the weights the drivers use for the edges between zones.

Another piece of information in the dataset is that time window targets for package delivery are included for a subset of the stops. These constraints are often trivially satisfied, and ignoring them altogether had minimal impact on the final score of our approach. This was also observed by other contestants of the Amazon Challenge [AA21; CHH22]. Therefore, time windows are ignored in our approach. Moreover, we also ignore



(a) Different colors represent different zones.

(b) Each zone is substituted by a *hypernode* containing all stops within it.

Figure 3.7: Example of an expert human route from the dataset in terms of its stop sequence and zone sequence.

all information about the size of the vehicle and the size of the packages to be delivered, as these do not seem to influence the routes chosen by the drivers.

3.3.2. COMPLETE METHOD

In this section, we outline all the steps involved in our IO approach to the Amazon Challenge. As explained in the previous section, due to the nature of the provided data, we focus on learning the preferences of the driver at the zone level. However, the historical routes of the datasets are given in terms of a sequence of stops. Moreover, given a new request for stops to be visited, the learned model should return the sequence of stops, not the sequence of zones. Therefore, intermediate steps need to be taken to go from a sequence of stops to a sequence of zones, and vice-versa. A block diagram of our method is shown in Figure 3.8. A detailed description of each step of our method is given in the following.

Step 1 (pre-process the data). The first step is to transform the datasets from stop-level information to zone-level information. Namely, for each data pair of stops to be visited $\hat{s}$ and respective expert route $\hat{x}$ (see R-TSP modeling in Section 3.2.3), we transform them into a signal $\hat{s}_t^z$ containing the zones to be visited and respective expert zone sequence $\hat{x}_t^z$. This is the process illustrated in Figure 3.7. However, differently from Figure 3.7a, there are cases in the dataset where the human driver visits a certain zone, leaves it, and later returns to the same zone. Thus, to enforce that the sequence of zones respects the TSP constraint that each zone is visited only once, when transforming a sequence of stops into a sequence of zones, we consider that a zone is visited at the time the *most consecutive stops* in that zone are visited. To illustrate it, consider the case when a driver visits 7 stops belonging to zones A, B, C, where the sequence of visited stops, in terms of their zones, is $A \to B \to B \to A \to A \to C \to C$. In this case, the driver visits zone A, leaves it, and then visits it again. Following our transformation rule, we consider the sequence of zones to be $B \to A \to C$.

Step 2 (Inverse Optimization). Next, considering the hypergraph of zones (i.e., each node represents a zone), we use our IO approach to learn the weights the expert drivers give to the edges connecting the zones. Namely, given a dataset of N examples of zones to be visited and respective zone sequences, we model the problem as an IO problem as

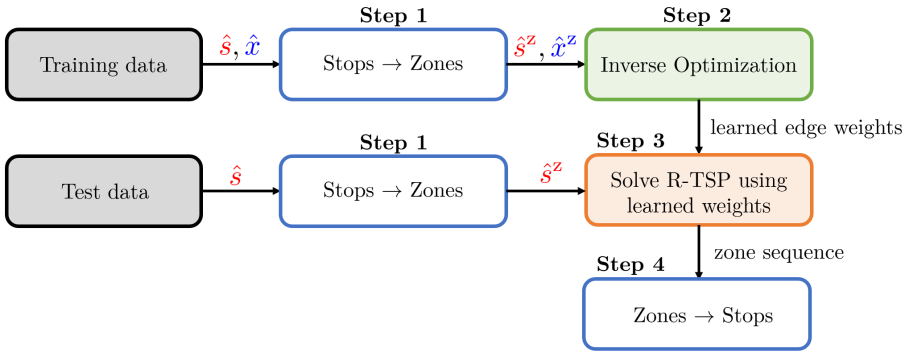


Figure 3.8: Overview of the proposed method.

in Section 3.2.3, and we solve Problem (3.3) using Algorithm 2 to learn a cost vector θ, that is, a vector with components corresponding to the learned edge weights between zones.

Step 3 (compute the zone sequence). Let $\hat{s}$ be a set of stops to be visited from the test dataset. To use the weights learned in Step 2 to construct a route for these stops, we first need to transform the signal from the stops to be visited into the *zones* that need to be visited by the driver $\hat{s}^z$ (Step 1). Given the signal of zones to be visited, and the weights θ learned in Step 2, we solve the R-TSP over zones with (3.12) as the cost function and (3.13) as the constraint set. Specific choices for M_{ij} will be discussed in Section 3.4.1. The solution to this problem contains the sequence of zones the driver needs to follow. In some cases, routes in the test dataset contain zones that are not visited in the training dataset. In these cases, since the vector of learned weights θ does not contain information about these zones, we set their weights equal to the Euclidean distance between the center of the zones.

Step 4 (from a zone sequence to a stop sequence). The final step of our method consists of computing the complete route at the stop level. In other words, given the zone sequence computed in Step 3 (e.g., Figure 3.7b), we want to find a respective stop sequence (e.g., Figure 3.7a). We do it using a penalization method. Let c_{ij} be the transit time from stop i to stop j (this information is provided in the Amazon Challenge dataset, see Figure 3.5). To enforce the zone sequence found in Step 3, we create the penalized weights $\tilde{c}_{ij}$, defined as

$$\tilde{c}_{ij} := \begin{cases} c_{ij}, & \text{if stops } i \text{ and } j \text{ are in the same zone} \\ c_{ij} + R, & \text{if the zone of stop } j \text{ should be visited directly after the zone of stop } i \\ c_{ij} + 2R, & \text{otherwise,} \end{cases}$$

where $R > 0$ is a penalization constant. For a large enough R, this modification ensures that all stops within a zone are visited before moving to another zone and that the sequence of zones from Step 3 is respected. Thus, we compute the complete route over a

set of stops $\hat{S}$ by solving the R-TSP over stops

$$\min_{x \in \mathbb{X}(\hat{S})} \sum_{i=1}^{m} \sum_{j=1}^{m} \tilde{c}_{ij} x_{ij},$$

where $\mathbb{X}$ is the R-TSP constraint set (3.13) and m is the total number of stops.

As explained in Section 3.3, the dataset comprises example routes from 5 cities in the USA, and each city can have multiple depots. It turns out that each zone is always served by the same depot, thus, we can learn the preferences of the drivers separately for each depot. Consequently, when using our approach in the Amazon Challenge, we perform steps 1 to 4 separately for each depot. More details on the number of zones served by each depot can be found in Section 3.4.2 and [Mer+22].

3.4. NUMERICAL RESULTS

In this section, we numerically evaluate our Inverse Optimization approach to the Amazon Challenge. To compute the zone sequence (i.e., step 3 of our method) we use a Gurobi-based TSP solver [Gur21] (except for the experiments in Section 3.4.2) and to compute the complete route at the stop-level (i.e., step 4 of our method), we use the LKH-3 solver [Hel17]. The difference between the two is that the Gurobi-based TSP solver is exact, but usually slower for large TSPs, whereas the LKH-3 solver is approximate, but usually faster. Thus, the choice of which solver to use is based on the size of the TSP problem that has to be solved. In our IO approach to the Amazon Challenge, the TSP problem over zones is usually a relatively small one (less than 50 zones), so we solve it using the exact Gurobi-based solver. On the other hand, the TSP problem over stops is usually a larger one (+100 stops), so we solve it using the LKH-3 solver (solving it using the Gurobi-based solver led to little to no improvement in the final Amazon score, while taking significantly more time). Our experiments are reproducible, and the underlying source code is available at [Zat23a]. In particular, we use the InvOpt python package [Zat23b] for the IO part of our approach.

3.4.1. IO FOR THE AMAZON CHALLENGE

In this section, we present results for two IO approaches: a general approach and the tailored approach proposed in this chapter. For both approaches, we use $\eta_t = 0.0005/t$ and θ_1 (that is, the initial point of the used algorithm) as the Euclidean distance between the center of the zones in the training dataset, where we compute the center of a zone by taking the mean of the longitudinal and lateral coordinates of all stops within the zone. All scores reported in this section are the Amazon score of the learned model evaluated in the test dataset.

General IO approach. As a benchmark for our tailored IO methodology, we apply a general IO methodology to the Amazon Challenge. This can be interpreted as using the general IO methodology from Chapter 2. In particular, we have the following design choices:

- *Hypothesis function:* We use the linear hypothesis (3.12) with $M_{ij} = 0 \; \forall i, j \in V$.
- *IO algorithm:* We use the SAMD algorithm with $T = 5N$, $\omega(\theta) = \frac{1}{2}\|\theta\|_2^2$, and $\Theta = \{\theta : \theta \geq 0\}$.

W	G	G	G	G	G	G	G	G	G	G	G	H	H	H	H	H
x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
y	1	2	3	2	1	1	2	3	3	2	1	1	2	3	3	2
Z	E	E	G	G	G	H	H	H	J	J	J	A	A	A	B	B

Table 3.1: Part of the zone sequence from the example route with RouteID f6cf991e-9bb0-46b9-a07d-8192c2d29bb1.

The final Amazon score of the learned IO model is **0.535**. This score ranks 11th compared to the 48 models that qualified for the final round of the Amazon Challenge [Ama21c]. Although this is already a good result, we can significantly improve this score by using our IO approach tailored to routing problems.

Tailored IO approach. To apply our tailored IO approach to the Amazon Challenge, we have the following design choices:

- *Hypothesis function:* We use the affine hypothesis (3.12). The weights M_{ij} of the affine term are defined below.
- *IO algorithm:* We use Algorithm 2, with standard update steps and $T = 5$.

As also noticed by some of the contestants of the original Amazon Challenge [WPN21], by carefully analyzing the sequence of zones followed by the human drivers, one can uncover patterns that can be exploited. These patterns are related to the specific encoding of the *Zone ID* given to the zones. Namely, the Amazon Zone IDs have the form *W-x.yZ*, where W and Z are upper-case letters and x and y are integers. Table 3.1 shows an example of a zone sequence from the Amazon Challenge dataset. Although the zone sequence shown in Table 3.1 is just a small example, it contains the patterns that we exploit to improve our approach, which are the following:

- *Area sequence*: For a zone with Zone ID *W-x.yZ*, define its *area* as *W-x.Z*. It is observed that the drivers tend to visit all zones within an area before moving to another area.
- *Region sequence*: For a zone with Zone ID *W-x.yZ*, define its *region* as *W-x*. It is observed that the drivers tend to visit all areas within a region before moving to the next region.
- *One unit difference:* Given two zone IDs $z_1 = W\text{-}x.yZ$ and $z_2 = A\text{-}b.cD$, we define the difference between two zone IDs as $d(z_1, z_2) := |\text{ord}(W) - \text{ord}(A)| + |x - b| + |y - c| + |\text{ord}(Z) - \text{ord}(D)|$, where the function `ord` maps characters to integers (in our numerical results, we use Python's built-in `ord` function). In particular, letters that come after the other in the alphabet are mapped to integers that differ by 1, e.g., ord(G) = 71 and ord(H) = 72. It is observed that for subsequent zone IDs in the zone sequences from the Amazon dataset, the difference between these zone IDs tends to be small (most often 1).

Next, we incorporate these observations into our IO learning approach. One way to force the routes from our IO model to respect these behaviors (i.e., the "area sequence",

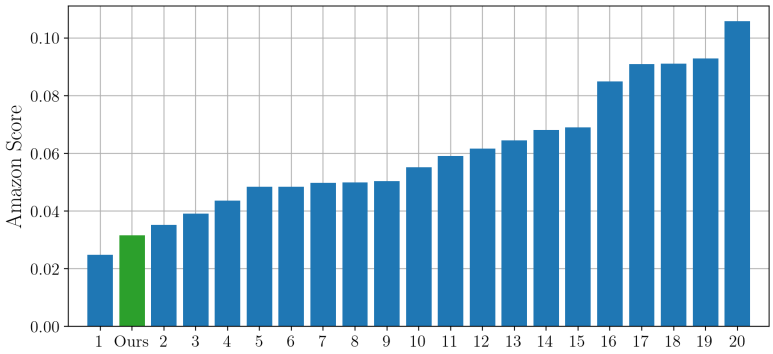


Figure 3.9: Our final score compared to the scores of the top 20 contestants of the Amazon Challenge.

"region sequence" and "one unit difference" behaviors) is to use penalization terms. In a sense, using these penalizations can be interpreted as modifying what we believe is the optimization problem the expert human drivers solve to compute their routes. Thus, we use (3.12) as our hypothesis function, with $M_{ij} = M^A_{ij} + M^R_{ij} + M^d_{ij}$, where $M^A_{ij} = 0$ if zones i and j are in the same area, and $M^A_{ij} = 1$ otherwise, $M^R_{ij} = 0$ if zones i and j are in the same region, and $M^R_{ij} = 1$ otherwise, and $M^d_{ij} = d(i, j)$, that is, the difference between zones i and j. Since for Algorithm 2 we initialize θ_1 as the Euclidean distance between zone centers, where the coordinates of the centers are given by their latitudes and longitudes, each component of θ_1 is much smaller than 1. This makes a penalization of one unit (such as the ones used for M^A_{ij} and M^R_{ij}) enough to enforce that the resulting routes will respect the area sequence and region sequence behaviors. The same idea applies to the "one unit variance" penalization.

The final Amazon Challenge score achieved by our tailored approach is **0.0302**, which significantly improves the 0.0535 score of the benchmark (i.e., general IO) approach. Figure 3.9 shows the scores of the top 20 submissions of the Amazon Challenge. As can be seen, our score ranks 2nd compared to the 48 models that qualified for the final round of the Amazon Challenge [Ama21c]. Compared to the initial weights fed to the tailored IO algorithm, considering only the set of weights changed by the first-order method, the change was 28.6% on average, with the 10th and 90th percentiles equal to 0.8% and 68.8%, respectively. These changes may be interpreted in the following sense: if the first-order algorithm increases the weight of the edge connecting zones A and B, it means that according to the data, the expert human driver considers this edge more costly than the initial weights (i.e., than the Euclidean distance between the zones), or in other words, the drivers have less preference in using this edge. Similarly, if the algorithm decreases the weight, we can interpret it as the drivers considering this edge less costly, thus, having a stronger preference in using this edge when driving. Moreover, to test the robustness of the learned model, we added Gaussian perturbations to the weights learned and computed the Amazon score of the perturbed model. Adding Gaussian perturbations with magnitudes (in expectation) of 0.1% and 1% compared to the average magnitude of the weight matrix led to an increase of 0.7% and 4% in the Amazon score, respectively.

Thus, we observed the expected behavior from a robust model: small perturbations lead to small changes in the model's output.

In our experience, small perturbations in the learned model do not tend to lead to significant changes in the resulting route. To evaluate its robustness, we can add Gaussian perturbations to the weights learned for the Amazon Challenge and compute their effect on the Amazon score of the model. Adding Gaussian perturbations with magnitudes (in expectation) of 0.1% and 1% compared to the average magnitude of the weight matrix led to an increase of 0.7% and 4% in the Amazon score, respectively. Thus, we observed the expected behavior from a robust model: small perturbations lead to small changes in the model's output while increasing the perturbations increases their impact on the model. We have added a discussion on this point to the revised version of the paper (page 21).

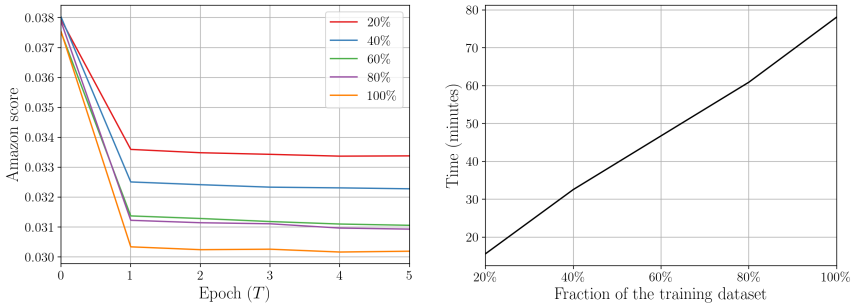
3.4.2. COMPUTATIONAL AND TIME COMPLEXITY

In this section, we present further numerical experiments using the Amazon Challenge datasets, focusing on the computational and time complexity of Algorithm 2. Before we present our results, as discussed at the end of Section 3.3.2, recall that we apply our IO learning method separately for each depot in the Amazon Challenge training dataset. Thus, assuming we can run Algorithm 2 in parallel for all depots, the complexity of computing the final IO model for all depots equals the complexity of computing the IO model for the largest depot in the dataset. For the Amazon Challenge, the largest depot dataset is DLA7 in Los Angeles, which we thus use to discuss the complexity of our approach.

Dataset size versus performance. First, we study the performance of our IO approach by changing the size of the training dataset. That is, instead of using the entire training dataset of the Amazon Challenge to train the IO model, we test the impact of using only a fraction of the available data. Figure 3.10 shows the results of this experiment. Figure 3.10a shows the Amazon score achieved, per epoch, by Algorithm 2 using different fractions of the Amazon training dataset. Figure 3.10b shows the time it took to run Algorithm 2 for 5 epochs, for the different fractions of the training dataset. As expected, the more data we feed to Algorithm 2, the better the score gets, and the longer the training takes. Interestingly, notice that using only 20% of the data provided for the challenge, our IO approach is already able to learn a routing model that scores 0.0334, which would still rank 2nd compared to the 48 models that qualified for the final round of the Amazon Challenge.

Time complexity and approximate A-FOP. In practice, the most time-consuming component of Algorithm 2 is solving the A-FOP (line 5). As previously explained, for the Amazon Challenge, this problem consists of a TSP over zones (see Step 2 in Section 3.3.2). Thus, for each epoch of Algorithm 2, we need to solve N TSPs, where N is the number of examples in the training dataset. For the depot DLA7, $N = 1133$, and each example contains, on average (rounded up), 23 zones, where the largest instance has 37 zones and the smallest has 9 zones. Thus, for each epoch of Algorithm 2, we need to solve 1133 TSPs, each with 23 zones on average. Using an exact Gurobi-based TSP solver, running 5 epochs of Algorithm 2 using the entire training dataset took 78.13 minutes (see Figure 3.10b).

However, recall that as discussed in Remark 3.2, Algorithm 2 can be used with an



(a) Amazon score on the test dataset, for models learned using different fractions of the training dataset.

(b) Time taken to run 5 epochs of Algorithm 2.

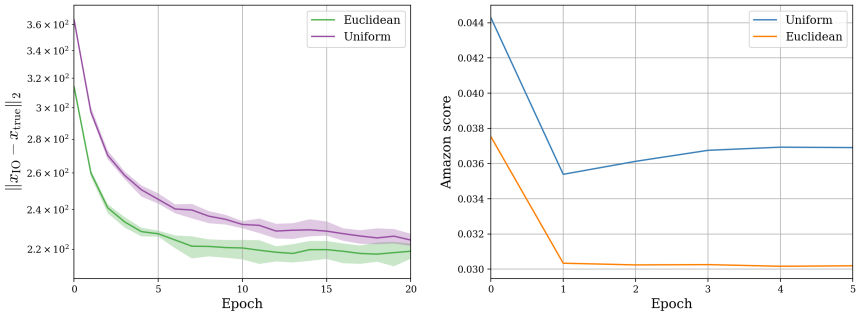
Figure 3.10: Comparison of the Amazon score and learning time of models using different fractions of the Amazon Challenge training dataset.

TSP solver	Gurobi	Gurobi	LKH-3	LKH-3	OR-Tools	OR-Tools
Dataset fraction (%)	20	100	20	100	20	100
Amazon score	0.0334	0.0302	0.0335	0.0302	0.0337	0.0306
Training time (min)	15.59	78.13	17.87	84.62	12.72	69.51

Table 3.2: Summary of the results of Section 3.4.2. The Amazon scores are computed using the test dataset. The TSP solver refers to the solver used to solve the A-FOP in line 5 of Algorithm 2, and the training time is the time of running Algorithm 2 for 5 epochs.

approximate A-FOP instead of an exact one. The idea here is that solving A-FOP approximately can be faster in practice, which may compensate for a potentially worse performance of the final learned IO model. We test this idea using Algorithm 2 with approximate TSP solvers instead of the exact Gurobi-based one. For the approximate solvers, we test the LKH-3 [Hel17] and Google OR-Tools [Goo21]. The final Amazon score after 5 epochs of Algorithm 2 using Google OR-Tools is 0.0306, just slightly worse compared to the Gurobi and LKH-3 solvers, but taking only 69.51 minutes in total. Interestingly, we can push this time even further. As can be seen in Figure 3.10a, a good IO model can be achieved using Algorithm 2 for only one epoch. Moreover, from Figure 3.10a, it can also be seen that a good IO model can be learned using only 20% of the training dataset. Thus, using 20% of the training dataset and running Algorithm 2 using the Google OR-Tools TSP solver for 5 epochs, we achieve a final score of 0.0337 (0.0341 after only one epoch) in only 12.72 minutes (i.e., 2.54 minutes per epoch on average). This showcases the learning efficiency of our IO methodology, making it also suitable for *real-time* applications, where models need to be learned/updated frequently, and the training time should not take more than a couple of minutes. Table 3.2 summarizes the numerical results of this section.

3.4.3. FURTHER NUMERICAL RESULTS



(a) Average error between the routes generated by θ_{true} and θ_{IO}.

(b) Amazon score of the learned models.

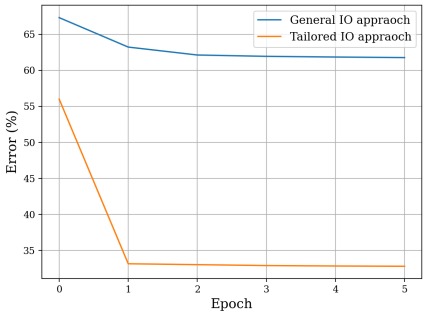
Figure 3.11: Comparison between different initializations of $\theta_1^{[1]}$ for Algorithm 2, for the VRPTW scenario of Section 3.2.2 and for the Amazon Challenge.

IMPACT OF THE INITIAL POINT

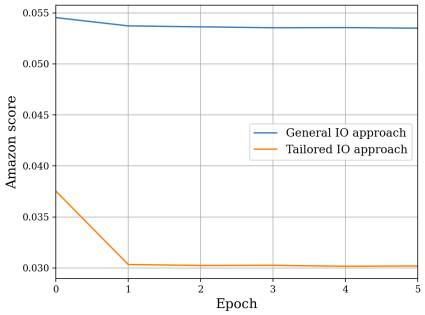
An important parameter of Algorithm 2 is the initial point $\theta_1^{[1]}$. In practice, the better the initial point, the faster the algorithm will converge, and perhaps more importantly, the better the test dataset performance of the final model tends to be. In this section, we investigate the impact of different choices of $\theta_1^{[1]}$ for the numerical experiment of Section 3.2.2 and for the Amazon Challenge. In particular, we compare the "Euclidean distance" initialization used to generate the results shown in Figure 3.3b and Figure 3.10a with a "uniform" initialization, where $\theta_1^{[1]}$ is a vector with all its components equal to the same number (this initialization could be used when no prior information on a good cost vector is known). Figure 3.11 shows the results of this experiment. As can be seen, using the Euclidean distance can accelerate the convergence of the algorithm, as in the VRPTW scenario, as well as improve the test dataset performance of the learned model, as in the case of the final Amazon score of the learned models for the Amazon Challenge. This means that, although the Euclidean weights do not explain the routes in the dataset, there is a correlation between the Euclidean distance between nodes and the true weights used to generate the observed routes.

ALTERNATIVE PERFORMANCE METRIC

In Section 3.4, we evaluated our results for the Amazon Challenge in terms of the Amazon score (see Section 3.3). In this section, we present results in terms of a zone sequence prediction error metric. Namely, given a zone sequence obtained from a learned IO model (i.e., the output of Step 3 of our IO approach, see Figure 3.8), and the zone sequence $\hat{x}$ from the training or test dataset, the prediction error Error$(x, \hat{x})$ counts how many zones in $\hat{x}$ are in the wrong position compared to x. For instance, if $x = \{$T-7.1C, T-7.1B, T-8.1B, T-8.1C, T-8.2C$\}$ and $\hat{x} = \{$T-7.1B, T-7.1C, T-8.1B, T-8.2C, T-8.1C$\}$, then the Error$(x, \hat{x}) = 4$. This performance measure can be interpreted as a generalization of the classical 0-1 error used for classification problems, where 0-1$(x, \hat{x}) = 0$ if $x = \hat{x}$, and 0-1$(x, \hat{x}) = 1$ otherwise. Thus, given a dataset of N examples of zone sequences and the respective sequences predicted by the IO model, we define the total (percentage) zone

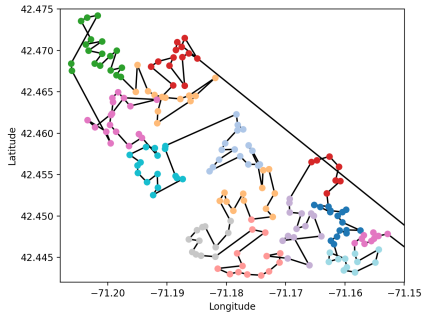


(a) Performance of the general and tailored IO approaches using the zone sequence prediction error.

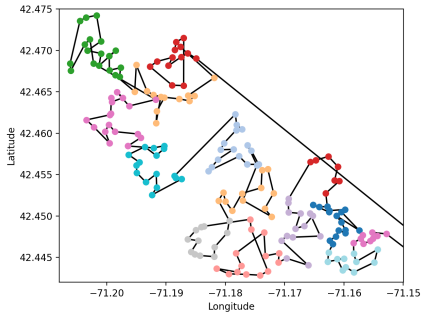


(b) Performance of the general and tailored IO approaches using the Amazon score.

Figure 3.12: Comparison between the zone sequence prediction error and Amazon score performance metrics.



(a) Route example from the test dataset.



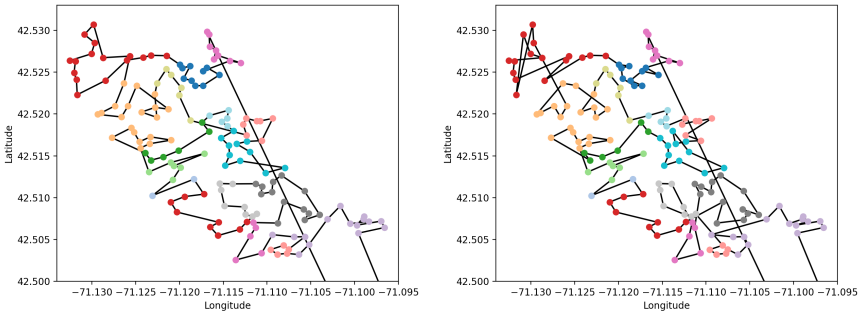
(b) Output route from the IO model.

Figure 3.13: Comparisson between the driver's route from the Amazon Challenge and the output of the IO model. In this example, the zone sequence predicted by the IO model perfectly matches the one from the original route.

sequence prediction error across the entire dataset as $100\sum_{i=1}^{N}\text{Error}(x^{[i]},\hat{x}^{[i]})/\sum_{i=1}^{N}L^{[i]}$, where $L^{[i]}$ is the length of the i'th zone sequence. In other words, this value can be interpreted as the percentage of time the IO approach correctly predicts the position of a zone in the zone sequence. Figure 3.12a shows the performance of the general and our tailored IO approaches from Section 3.4.1 in terms of the zone sequence prediction error. For comparison, we also show their respective Amazon score in Figure 3.12b. As can be seen, the IO models show (qualitatively) similar performance, in terms of both prediction error and Amazon score metrics.

ROUTE EXAMPLES

In this section, we show some route examples, comparing the routes of human drivers from the Amazon Challenge dataset, with the routes from our IO approach. In Figure 3.13, we show an example where the zone sequence predicted by the IO model (i.e., Step 3 in Section 3.3.2) perfectly matches the one from the original route, where nodes of different colors represent different zones. As can be noticed, even though the zone se-



(a) Route example from the test dataset.

(b) Output route from the IO model.

Figure 3.14: Comparisson between the driver's route from the Amazon Challenge and the output of the IO model. In this example, the zone sequence predicted by the IO model does not match the one from the original route.

quence is the same, the sequence of stops within each zone is different. However, even with these differences, the Amazon score of the route in Figure 3.13b is still quite small (0.0046). This phenomenon is generally observed for the Amazon Challenge: perfectly predicting the zone sequence tends to lead to a small Amazon score, even with different sequences of stops within each zone. This observation supports our IO approach to the challenge, where we focused on predicting the correct zone sequence, instead of the stop sequences.

In Figure 3.14 we show an example where the zone sequence from the original route (Figure 3.14a) differs from the one predicted by the IO model (Figure 3.14b). In particular, the zone prediction error (defined in Section 3.4.3) between these two routes is 31.6%. Still, since the zones predicted in the wrong order are close to each other, the Amazon score of the route in Figure 3.14b is relatively small (0.0117). This example provides some intuition on the results from Figure 3.12: even though the average zone prediction error of the proposed tailored IO approach is around 32%, the fact that it still guarantees a low Amazon score means that even when predicting the wrong zone sequence, the predicted zones in general similar (i.e., geographically close) to the actual ones from the test dataset.

4

CONCLUSION

In this thesis, we have studied the theory of IO, in terms of geometrical understanding, tractable reformulations, loss minimization interpretations, and efficient first-order algorithms. We also developed tailored IO methodologies to solve learning in routing problems and showed its real-world potential through the Amazon Challenge. A more detailed summary of our contributions is as follows.

4.1. SUMMARY

In Chapter 2, we have presented new approaches to tackle IO problems. Based on the geometry of the set of consistent cost vectors, we proposed the use of an incenter vector of this set as the solution to the IO problem. Moreover, we proposed a new loss function for IO problems: the Augmented Suboptimality Loss. This loss can be interpreted as a relaxation of the incenter approach to tackle problems with inconsistent data. Moreover, this loss can be used to derive IO approaches tailored for problems with mixed-integer feasible sets and can be optimized directly using first-order methods. For the latter case, we proposed new algorithms that combine stochastic, approximate, and mirror descent updates, all of which can be used to exploit the structure of IO losses. Finally, we numerically evaluated the proposed IO approaches and show that they can outperform state-of-the-art methods in a variety of scenarios.

In Chapter 3, we have proposed an IO methodology for learning the preferences of decision-makers in routing problems. To exemplify the potential and flexibility of our approach, we first applied it to a simple CVRP problem, where we gave insight into how our IO algorithm works by modifying the learned edge weights by comparing the example routes to the optimal route we get using the current learned weights. Then, we applied it to a larger VRPTW example, comparing the performance of our proposed algorithm with different approaches from the literature. Finally, we have shown the real-world potential of our approach by using it to tackle the Amazon Challenge, where the goal of the challenge was to develop routing models that replicate the behavior of real-world expert human drivers. To do so, we first defined what we call Restricted TSPs (i.e.,

TSPs for which only a subset of the nodes is required to be visited). Given a dataset of signals (nodes to be visited) and expert responses (R-TSPs tours), we have shown how to use IO to learn the edge weights that explain the observed data. In the context of the Amazon Challenge, learning these edge weights translates to learning the sequence of city zones preferred by expert human drivers. Then, from a sequence of zones, we constructed a complete TSP tour over the required stops. The final score of our approach is **0.0302**, which ranks 2nd compared to the 48 models that qualified for the final round of the Amazon Challenge.

4.2. REFLECTIONS AND FUTURE RESEARCH DIRECTIONS

The work presented in this thesis opens several avenues for future research.

Geometry, optimization algorithms, and structured prediction. One could try different ways to choose a vector from the set of consistent costs, for example, using ellipsoidal cones instead of circular cones. This idea was used in online IO algorithms (see [BFL23, Section 4.3]), but perhaps it can also be leveraged for better empirical performance in the offline case. Also, one could try to adapt and further develop the incenter approaches presented in this thesis to the online IO scenario, perhaps leveraging the geometry of the IO problem to prove tighter regret bounds in some specific scenarios, akin to [BFL23]. Another idea is to come up with different algorithms to tackle the optimization problems discussed in this thesis, for instance, using cutting-plane/bundle methods, extending the related literature of structured prediction problems [Wan09; JFY09; Teo+10]. Moreover, given the wealth of different first-order methods in the literature, we feel like designing specialized first-order (or even proximal or second-order) algorithms tailored for IO problems is an under-explored research direction. Given the newly established bridge between IO and structured prediction problems (see Remark 2.10), we expect that many interesting results can be achieved by combining and extending the literature of these two, mostly disjoint, communities.

Applications to routing/combinatorial problems. Regarding the application of IO to routing problems, it would be interesting to apply our methodology to different and more complex classes of routing problems, for instance, dynamic VRPs, routing problems with backhauls, as well as routing problems with continuous decision variables. Moreover, although in this work we focused on routing problems, our methodology could also be adapted and tailored to different classes of problems with a binary decision space, such as 0-1 knapsack problems. Given the modularity/flexibility of our IO methodology, we believe it has the potential to be used for a wide range of real-world decision-making problems.

Open-source IO software. A particularly impactful work direction is further to develop the InvOpt python package for IO [Zat23b]. As of the moment of the writing of this thesis, InvOpt is still in the development stage and can be greatly improved, e.g., by implementing more IO approaches and replicating numerical results from the literature. Also, the package can be made more robust by using professional software development tools.

Currently, the literature of IO is still in constant development and change, thus, having an open-source implementation of IO algorithms and methods could be of great use to the IO community.

A priori guarantees. For the most part, the evaluation metrics used in the IO literature to test the quality of the learned models are based on a posteriori out-of-sample tests (e.g., the results from Section 2.4). This is common practice for supervised learning problems (such as IO), and reflects the idea that we want the learned model to generalize to unseen data. However, an under-explored area in the IO literature is on *a priori* guarantees, that is, looking only at the data and learning method, what guarantees can we give about the learned model? One possible way to tackle this question is to interpret the IO problem as a *System Identification* (SysId) problem. For SysId problems, a classical concept that allows us to prove a priori results on the learned (i.e., identified) model is the so-called *persistency of excitation* of the data. Thus, an interesting research direction is to develop persistency of excitation conditions in the context of IO data, which allows us to prove a priori guarantees on the quality of IO methods and models.

IO and Discrete Choice Modelling. In the literature on modeling discrete decision-making processes, a popular approach is to use so-called *Discrete Choice Models* (DCM). These models originate from the econometrics literature and have their roots in *random utility theory*. This differs from standard IO models, which are based on deterministic forward optimization problems. Despite this difference, many parallels can be drawn between DCM and IO, e.g., both can be interpreted as methods that tackle these modeling problems by choosing a hypothesis class (i.e., cost/utility functions), and optimize a loss function that computes the likelihood/suboptimality of a model. Thus, a possibly fruitful research duration is to investigate the connection between these two approaches, bridging the DCM and IO communities, which are currently mostly unaware of each other's literature.

BIBLIOGRAPHY

[AA21] Okan Arslan and Rasit Abay. “Data-Driven Vehicle Routing in Last-Mile Delivery”. In: *Technical Proceedings of the Amazon Last Mile Routing Research Challenge*. dspace.mit.edu. 2021.

[AG05] Shabbir Ahmed and Yongpei Guan. “The inverse optimal value problem”. In: *Mathematical Programming* 102 (2005), pp. 91–110.

[AKE21] Syed Adnan Akhtar, Arman Sharifi Kolarijani, and Peyman Mohajerin Esfahani. “Learning for control: An inverse optimization approach”. In: *IEEE Control Systems Letters* 6 (2021), pp. 187–192.

[Ama21a] Amazon.com, Inc. *Amazon Last Mile Routing Research Challenge*. URL: https://routingchallenge.mit.edu/. Last visited on 2022/10/28. 2021.

[Ama21b] Amazon.com, Inc. *Amazon Routing Challenge: scoring*. URL: https://github.com/MIT-CAVE/rc-cli/tree/main/scoring. Last visited on 2023/05/20. 2021.

[Ama21c] Amazon.com, Inc. *Last-Mile Routing Challenge Team Performance and Leaderboard*. URL: https://routingchallenge.mit.edu/last-mile-routing-challenge-team-performance-and-leaderboard/. Last visited on 2022/10/28. 2021.

[AO01] Ravindra K. Ahuja and James B. Orlin. “Inverse optimization”. In: *Operations Research* 49.5 (2001), pp. 771–783.

[AO14] Zeyuan Allen-Zhu and Lorenzo Orecchia. “Linear coupling: An ultimate unification of gradient and mirror descent”. In: *arXiv preprint arXiv:1407.1537* (2014).

[ASS18] Anil Aswani, Zuo-Jun Shen, and Auyon Siddiq. “Inverse optimization with noisy data”. In: *Operations Research* 66.3 (2018), pp. 870–892.

[BCZ22] Merve Bodur, Timothy C. Y. Chan, and Ian Yihang Zhu. “Inverse mixed integer optimization: Polyhedral insights and trust region methods”. In: *INFORMS Journal on Computing* 34.3 (2022), pp. 1471–1488.

[Ber08] Dimitri P. Bertsekas. *Nonlinear programming*. Athena Scientific, 2008.

[Ber15] Dimitri P. Bertsekas. *Convex optimization algorithms*. Athena Scientific, 2015.

[BFL23] Omar Besbes, Yuri Fonseca, and Ilan Lobel. “Contextual inverse optimization: Offline and online learning”. In: *Operations Research* (2023).

[BGP12] Dimitris Bertsimas, Vishal Gupta, and Ioannis Ch. Paschalidis. “Inverse optimization: A new perspective on the Black-Litterman model”. In: *Operations Research* 60.6 (2012), pp. 1389–1403.

[BGP15] Dimitris Bertsimas, Vishal Gupta, and Ioannis Ch. Paschalidis. “Data-driven estimation in equilibrium using inverse optimization”. In: *Mathematical Programming* 153.2 (2015), pp. 595–633.

[BPS17] Andreas Bärmann, Sebastian Pokutta, and Oskar Schneider. “Emulating the expert: Inverse optimization through online learning”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 400–410.

[BT92] Didier Burton and Ph L. Toint. “On an instance of the inverse shortest paths problem”. In: *Mathematical Programming* 53 (1992), pp. 45–61.

[Bub15] S. Bubeck. *Convex Optimization: Algorithms and Complexity*. Foundations and Trends in Machine Learning. Now Publishers, 2015.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[Can+21] Rocsildes Canoy et al. “Learn-n-Route: Learning implicit preferences for vehicle routing”. In: *arXiv preprint arXiv:2101.03936* (2021).

[Can+24] Rocsildes Canoy et al. “Probability estimation and structured output prediction for learning preferences in last mile delivery”. In: *Computers & Industrial Engineering* (2024), p. 109932.

[CCZ14] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*. Graduate Texts in Mathematics. Springer International Publishing, 2014. ISBN: 9783319110080.

[CD12] Yerim Chung and Marc Demange. “On inverse traveling salesman problems”. In: *4OR* 10.2 (2012), pp. 193–209.

[CG19] Rocsildes Canoy and Tias Guns. “Vehicle routing by learning from historical solutions”. In: *Principles and Practice of Constraint Programming: 25th International Conference, CP 2019, Stamford, CT, USA, September 30–October 4, 2019, Proceedings 25*. Springer. 2019, pp. 54–70.

[Cha+14] Timothy C. Y. Chan et al. “Generalized inverse multiobjective optimization with application to cancer therapy”. In: *Operations Research* 62.3 (2014), pp. 680–695.

[Cha+22] Timothy C. Y. Chan et al. “An inverse optimization approach to measuring clinical pathway concordance”. In: *Management Science* 68.3 (2022), pp. 1882–1903.

[CHH22] William Cook, Stephan Held, and Keld Helsgaun. “Constrained local search for last-mile routing”. In: *Transportation Science* (2022).

[CK20] Violet Xinying Chen and Fatma Kılınç-Karzan. “Online Convex Optimization Perspective for Learning from Dynamically Revealed Preferences”. In: *arXiv preprint arXiv:2008.10460* (2020).

[CL21] Ivan Contreras and Dang Le. “Learning Routing Models with Cluster Precedence Constraints”. In: *Technical Proceedings of the Amazon Last Mile Routing Research Challenge*. dspace.mit.edu. 2021.

[CLT19] Timothy C. Y. Chan, Taewoo Lee, and Daria Terekhov. “Inverse optimization: Closed-form solutions, geometry, and goodness of fit”. In: *Management Science* 65.3 (2019), pp. 1115–1135.

[CMB21] Rocsildes Canoy, Jayanta Mandi, and Victor Bucarey. “TSP with learned zone preferences for last mile vehicle dispatching”. In: *Technical Proceedings of the Amazon Last Mile Routing Research Challenge*. dspace.mit.edu. 2021.

[CMZ23] Timothy C. Y. Chan, Rafid Mahmood, and Ian Yihang Zhu. “Inverse optimization: Theory and applications”. In: *Operations Research* (2023).

[CR12] Joseph Y. J. Chow and Will W. Recker. “Inverse optimization with endogenous arrival time constraints to calibrate the household activity pattern problem”. In: *Transportation Research Part B: Methodological* 46.3 (2012), pp. 463–479.

[CRJ14] Joseph Y. J. Chow, Stephen G. Ritchie, and Kyungsoo Jeong. “Nonlinear inverse optimization for parameter estimation of commodity-vehicle-decoupled freight assignment”. In: *Transportation Research Part E: Logistics and Transportation Review* 67 (2014), pp. 71–91.

[DFJ54] George Dantzig, Ray Fulkerson, and Selmer Johnson. “Solution of a large-scale traveling-salesman problem”. In: *Journal of the Operations Research Society of America* 2.4 (1954), pp. 393–410.

[DG17] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[DW11] Zhaoyang Duan and Lizhi Wang. “Heuristic algorithms for the inverse mixed integer linear programming problem”. In: *Journal of Global Optimization* 51 (2011), pp. 463–471.

[EG22] Adam N. Elmachtoub and Paul Grigas. “Smart “predict, then optimize””. In: *Management Science* 68.1 (2022), pp. 9–26.

[El +19] Othman El Balghiti et al. “Generalization Bounds in the Predict-then-Optimize Framework”. In: *Advances in Neural Information Processing Systems* (2019).

[FFK13] Mogens Fosgerau, Emma Frejinger, and Anders Karlstrom. “A link based network route choice model with unrestricted choice set”. In: *Transportation Research Part B: Methodological* 56 (2013), pp. 70–80.

[FSS03] András Faragó, Áron Szentesi, and Balázs Szviatovszki. “Inverse optimization in high-speed networks”. In: *Discrete Applied Mathematics* 129.1 (2003), pp. 83–98.

[Gho+18] Kimia Ghobadi et al. “Robust inverse optimization”. In: *Operations Research Letters* 46.3 (2018), pp. 339–344.

[GM21] Kimia Ghobadi and Houra Mahmoudzadeh. “Inferring linear feasible regions using inverse optimization”. In: *European Journal of Operational Research* 290.3 (2021), pp. 829–843.

[GMW21] Xiaotong Guo, Baichuan Mo, and Qingyi Wang. “Last-Mile Delivery Trajectory Prediction Using Hierarchical TSP with Customized Cost Matrix”. In: *Technical Proceedings of the Amazon Last Mile Routing Research Challenge*. dspace.mit.edu. 2021.

[Goo21] Google. *Traveling Salesperson Problem*. URL: https://developers.google.com/optimization/routing/tsp. Last visited on 2022/10/28. 2021.

[Gur21] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2021. URL: https://www.gurobi.com.

[Hel17] Keld Helsgaun. “An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems”. In: *Roskilde University* (2017).

[Heu04] Clemens Heuberger. “Inverse combinatorial optimization: A survey on problems, methods, and results”. In: *Journal of Combinatorial Optimization* 8 (2004), pp. 329–361.

[HK10] Alan J. Hoffman and Joseph B. Kruskal. “Integral boundary points of convex polyhedra”. In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art* (2010), pp. 49–76.

[IK05] Garud Iyengar and Wanmo Kang. “Inverse conic programming with applications”. In: *Operations Research Letters* 33.3 (2005), pp. 319–330.

[JFY09] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. “Cutting-plane training of structural SVMs”. In: *Machine Learning* 77.1 (2009), pp. 27–59.

[JN11] Anatoli Juditsky and Arkadi Nemirovski. “First order methods for nonsmooth convex large-scale optimization, i: general purpose methods”. In: *Optimization for Machine Learning* 30.9 (2011), pp. 121–148.

[KW97] Jyrki Kivinen and Manfred K. Warmuth. “Exponentiated gradient versus gradient descent for linear predictors”. In: *Information and Computation* 132.1 (1997), pp. 1–63.

[KWB11] Arezou Keshavarz, Yang Wang, and Stephen Boyd. “Imputing a convex objective function”. In: *2011 IEEE International Symposium on Intelligent Control*. 2011, pp. 613–619. DOI: 10.1109/ISIC.2011.6045410.

[LFN18] Haihao Lu, Robert M. Freund, and Yurii Nesterov. “Relatively smooth convex optimization by first-order methods, and applications”. In: *SIAM Journal on Optimization* 28.1 (2018), pp. 333–354.

[Liu+20] Shan Liu et al. “Integrating Dijkstra’s algorithm into deep inverse reinforcement learning for food delivery route planning”. In: *Transportation Research Part E: Logistics and Transportation Review* 142 (2020), p. 102070.

[LM01] Yuh-Jye Lee and Olvi L. Mangasarian. “SSVM: A smooth support vector machine for classification”. In: *Computational Optimization and Applications* 20.1 (2001), pp. 5–22.

[Lon+24] Youyuan Long et al. “Scalable Kernel Inverse Optimization”. In: *Advances in Neural Information Processing Systems* (2024), submmited.

[LSB12] Simon Lacoste-Julien, Mark Schmidt, and Francis Bach. "A simpler approach to obtaining an O (1/t) convergence rate for the projected stochastic subgradient method". In: *arXiv preprint arXiv:1212.2002* (2012).

[Mer+22] Daniel Merchán et al. "2021 Amazon Last Mile Routing Research Challenge: Data Set". In: *Transportation Science* (2022).

[MKR20] Konstantin Mishchenko, Ahmed Khaled, and Peter Richtárik. "Random reshuffling: Simple analysis with vast improvements". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 17309–17320.

[Moh+18a] Peyman Mohajerin Esfahani et al. "Data-driven inverse optimization with imperfect information". In: *Mathematical Programming* 167.1 (2018), pp. 191–234.

[Moh+18b] Peyman Mohajerin Esfahani et al. "Data-driven inverse optimization with imperfect information". In: *Mathematical Programming* 167.1 (2018), pp. 191–234.

[Mor93] Jorge J. Moré. "Generalizations of the trust region problem". In: *Optimization Methods and Software* 2.3-4 (1993), pp. 189–209.

[Nem04] Arkadi Nemirovski. "Prox-method with rate of convergence O(1/t) for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems". In: *SIAM Journal on Optimization* 15.1 (2004), pp. 229–251.

[Nem96] Arkadi Nemirovski. *Lecture Notes: Interior point polynomial methods in convex programming*. URL: https://www2.isye.gatech.edu/~nemirovs/Lect_IPM.pdf. Last visited on 2022/06/23. 1996.

[Nes07] Yurii Nesterov. "Dual extrapolation and its applications to solving variational inequalities and related problems". In: *Mathematical Programming* 109.2-3 (2007), pp. 319–344.

[NL+11] Sebastian Nowozin, Christoph H. Lampert, et al. "Structured learning and prediction in computer vision". In: *Foundations and Trends® in Computer Graphics and Vision* 6.3–4 (2011), pp. 185–365.

[Ora19] Francesco Orabona. "A modern introduction to online learning". In: *arXiv preprint arXiv:1912.13213* (2019).

[ORT22] ORTEC. *EURO Meets NeurIPS 2022 Vehicle Routing Competition*. URL: https://euro-neurips-vrp-2022.challenges.ortec.com/. Last visited on 2023/01/24. 2022.

[Pit21] Anselmo R. Pitombeira-Neto. "Route sequence prediction through inverse reinforcement learning". In: *Technical Proceedings of the Amazon Last Mile Routing Research Challenge*. dspace.mit.edu. 2021.

[PW14] Ting Kei Pong and Henry Wolkowicz. "The generalized trust region subproblem". In: *Computational Optimization and Applications* 58.2 (2014), pp. 273–322.

[RBZ06] Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. “Maximum margin planning”. In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 729–736.

[RBZ07] Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. “(Approximate) Subgradient methods for structured prediction”. In: *Artificial Intelligence and Statistics*. PMLR. 2007, pp. 380–387.

[Sah74] Sartaj Sahni. “Computationally related problems”. In: *SIAM Journal on Computing* 3.4 (1974), pp. 262–279.

[Sch09] Andrew J. Schaefer. “Inverse integer programming”. In: *Optimization Letters* 3 (2009), pp. 483–489.

[Sho85] Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*. Springer Series in Computational Mathematics. Springer, 1985.

[SM17] Javier Saez-Gallego and Juan M. Morales. “Short-term forecasting of price-responsive loads using inverse optimization”. In: *IEEE Transactions on Smart Grid* 9.5 (2017), pp. 4805–4814.

[SMW95] W. Nick Street, Olvi L. Mangasarian, and William H. Wolberg. “An inductive learning approach to prognostic prediction”. In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 522–530.

[SSC21] Jianhan Song, Ashutosh Shukla, and Josiah Coad. “Inverse Reinforcement Learning for Learning Driver Utility Function for Package Delivery Routing Problems”. In: *Technical Proceedings of the Amazon Last Mile Routing Research Challenge*. dspace.mit.edu. 2021.

[Tas+05] Ben Taskar et al. “Learning structured prediction models: A large margin approach”. In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 896–903.

[Tas+06] Ben Taskar et al. “Structured Prediction, Dual Extragradient and Bregman Projections.” In: *Journal of Machine Learning Research* 7.7 (2006).

[Teo+10] Choon Hui Teo et al. “Bundle Methods for Regularized Risk Minimization”. In: *Journal of Machine Learning Research* 11.1 (2010).

[Tib96] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[TSK22] Bahar Taşkesen, Soroosh Shafieezadeh-Abadeh, and Daniel Kuhn. “Semi-discrete optimal transport: Hardness, regularization and numerical solution”. In: *Mathematical Programming* (2022), pp. 1–74.

[Tso+05] Ioannis Tsochantaridis et al. “Large margin methods for structured and interdependent output variables.” In: *Journal of Machine Learning Research* 6.9 (2005).

[TV02] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.

[Wan09] Lizhi Wang. “Cutting plane algorithms for the inverse mixed integer linear programming problem”. In: *Operations Research Letters* 37.2 (2009), pp. 114–116.

[WK22] Alex L Wang and Fatma Kılınç-Karzan. “The generalized trust region subproblem: solution complexity and convex hull results”. In: *Mathematical Programming* 191.2 (2022), pp. 445–486.

[WLK24] Niels A. Wouda, Leon Lan, and Wouter Kool. “PyVRP: a high-performance VRP solver package”. In: *INFORMS Journal on Computing* (2024). DOI: 10.1287/ijoc.2023.0055. URL: https://doi.org/10.1287/ijoc.2023.0055.

[WPN21] Matthias Winkenbach, Steven Parks, and Joseph Noszek. “Technical Proceedings of the Amazon Last Mile Routing Research Challenge”. In: *dspace.mit.edu* (2021).

[Wu+22] Chen Wu et al. “Learning from Drivers to Tackle the Amazon Last Mile Routing Research Challenge”. In: *arXiv preprint arXiv:2205.04001* (2022).

[Wul+17] Markus Wulfmeier et al. “Large-scale cost function learning for path planning using deep inverse reinforcement learning”. In: *The International Journal of Robotics Research* 36.10 (2017), pp. 1073–1087.

[Xu+18] Susan Jia Xu et al. “Network learning via multiagent inverse transportation problems”. In: *Transportation Science* 52.6 (2018), pp. 1347–1364.

[ZAM24] Pedro Zattoni Scroccaro, Bilge Atasoy, and Peyman Mohajerin Esfahani. “Learning in Inverse Optimization: Incenter Cost, Augmented Suboptimality Loss, and Algorithms”. In: *Operations Research* (2024), forthcoming.

[Zat+24] Pedro Zattoni Scroccaro et al. “Inverse Optimization for Routing Problems”. In: *Transportation Science* (2024), https://pubsonline.informs.org/doi/abs/10.1287/trsc.2023.0241.

[Zat23a] Pedro Zattoni Scroccaro. *Inverse Optimization for the Amazon Challenge*. https://github.com/pedroszattoni/amazon-challenge. 2023.

[Zat23b] Pedro Zattoni Scroccaro. *InvOpt: An open-source Python package to solve Inverse Optimization problems*. https://github.com/pedroszattoni/invopt. 2023.

[ZKM23] Pedro Zattoni Scroccaro, Arman Sharifi Kolarijani, and Peyman Mohajerin Esfahani. “Adaptive composite online optimization: Predictions in static and dynamic environments”. In: *IEEE Transactions on Automatic Control* 68.5 (2023), pp. 2906–2921.

[ZMA06] Elias Zafiropoulos, Ilias Maglogiannis, and Ioannis Anagnostopoulos. “A support vector machine approach to breast cancer diagnosis and prognosis”. In: *IFIP international conference on artificial intelligence applications and innovations*. Springer. 2006, pp. 500–507.

ACKNOWLEDGEMENTS

Four years ago, I started my PhD with the goal of deepening my knowledge about optimization, algorithms, machine learning, and mathematics in general. Looking back, I feel like I achieved my goal. What I did not expect, was that I would end up learning so much more. I learned about collaboration, negotiation, time management, project management, and how to present my ideas in the most effective way possible, all of which will serve me well, independently on the path I choose in my professional career.

Of course, none of it would have been possible without the right people and an appropriate learning environment. For this, I must thank my two supervisors: Bilge and Peyman. Throughout my PhD experience, they helped me with their guidance, knowledge, honesty, and as an endless source of inspiration. I would also like to thank all other colleagues from TU Delft and elsewhere with witch, at some point, I interacted, and thus, little by little, helped create the professional I am today.

Finally, I would like to thank my friends and family, who gave me unconditional support, and ultimately are also, directly and indirectly, responsible for the success of my PhD journey.

Pedro Zattoni Scroccaro
Delft, April 2024

CURRICULUM VITÆ

Pedro ZATTONI SCROCCARO

20-12-1993 Born in Curitiba, Brazil.

EDUCATION

2011–2017 B.Eng. Control and Automation Engineering
Federal University of Technology – Paraná

2018–2020 M.Sc. Systems and Control, *cum laude*
Delft University of Technology
Thesis: Online Convex Optimization with Predictions
Promotor: Dr. P. Mohajerin Esfahani

2020–2024 Ph.D.
Delft University of Technology
Thesis: Inverse Optimization Theory and Applications to Routing Problems
Promotors: Dr. B. Atasoy & Dr. P. Mohajerin Esfahani

List of Publications

1. Learning in Inverse Optimization: Incenter Cost, Augmented Suboptimality Loss, and Algorithms
   Pedro Zattoni Scroccaro, Bilge Atasoy, Peyman Mohajerin Esfahani
   Operations Research, forthcoming, 2024

2. Inverse Optimization for Routing Problems
   Pedro Zattoni Scroccaro, Piet van Beek, Peyman Mohajerin Esfahani, Bilge Atasoy
   Transportation Science, 2024, https://pubsonline.informs.org/doi/abs/10.1287/trsc.2023.0241

3. Adaptive Composite Online Optimization: Predictions in Static and Dynamic Environments
   Pedro Zattoni Scroccaro, Arman Sharifi Kolarijani, Peyman Mohajerin Esfahan
   IEEE Transactions on Automatic Control, vol. 68, no. 5, pp. 2906-2921, 2023