University of California
Santa Barbara

Bilevel Optimization in Learning and Control with Applications to Network Flow Estimation

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Mechanical Engineering

by

Francesco Seccamonte

December 2023

The Dissertation of Francesco Seccamonte is approved.

Professor Jeff Moehlis

Professor Ambuj K. Singh

Professor Enoch Yeung

Professor Francesco Bullo, Committee Chair

September 2023

Bilevel Optimization in Learning and Control with Applications to Network Flow Estimation

Ai miei genitori

Acknowledgements

I would like to express my gratitude to my adviser Prof. Francesco Bullo, for welcoming me to his lab and providing priceless guidance throughout my doctoral studies. I value how Francesco granted me substantial academic freedom, and at the same time showed constant support and encouragement. Special thanks also to Prof. Ambuj K. Singh, for serving as my unofficial co-adviser during the second half of my Ph.D. and giving me extremely useful feedback on the more learning-inclined parts of my thesis.

Even though I have experienced a somewhat limited Ph.D. due to the pandemic, I am thankful to all the fellow graduate students and researchers I have crossed paths with, in Francesco's group and more broadly at UCSB.

I am thankful to my committee members, Prof. Jeff Moehlis and Prof. Enoch Yeung, for their feedback on my dissertation research and the excitement expressed during my talks. I would also like to thank Dr. Ananthram Swami, for the interesting conversations we had while working on the DTRA project.

My internship at Motional exposed me to a dynamic and collaborative environment, where I worked with extremely smart and stimulating colleagues while "taking a break" from my dissertation research. I am thankful for the opportunity I was given.

I have been fortunate enough to meet amazing people in every place I have lived in: Starting with my hometown friends, to the Pavia group, to the Zürich one, all the way to the friends I have made in Singapore, Boston, and lately in California. I don't need to name names: Despite being scattered all over the world, your continued support has been (and continues to be) fundamental to my well-being, and made me always feel loved.

Lastly, I wish to thank my family for their tireless support: ten thousands kilometers have not been enough to loosen our bond. Your love has allowed me to survive the roller coaster ride I have been on these past years, and just knowing you are always there for

me means the world to me.

My research was partially supported by the US Defense Threat Reduction Agency grant HDTRA1-19-1-0017, and by the US Army Engineer Research and Development Center grant W912HZ-22-2-0010.

Curriculum Vitæ

Francesco Seccamonte

Education

2023	**Ph.D., Mechanical Engineering** University of California, Santa Barbara
2017	**M.Sc., Robotics, Systems and Control** ETH Zürich
2015	**B.Sc., Industrial Engineering** Universitá degli studi di Pavia

Peer-reviewed Academic Publications

J1 A. Carron, F. Seccamonte, C. Ruch, E. Frazzoli, M. N. Zeilinger. Scalable Model Predictive Control for Autonomous Mobility on Demand Systems. *IEEE Transactions on Control Systems Technology*, 2019. doi:10.1109/TCST.2019.2954520

C3 F. Seccamonte, A. K. Singh, F. Bullo. Inference of Infrastructure Network Flows via Physics-Inspired Implicit Neural Networks. *IEEE Conference on Control Technologies and Applications (CCTA)*, Bridgetown, Barbados, 2023.

C2 K. D. Smith, F. Seccamonte, A. Swami, F. Bullo. Physics-Informed Implicit Representations of Equilibrium Network Flows. *2022 Advances in Neural Information Processing Systems*.

C1 F. Seccamonte, J. Kabzan, E. Frazzoli. On Maximizing Lateral Clearance of an Autonomous Vehicle in Urban Environments. *2019 IEEE ITSC Intelligent Transportation Systems Conference*, Auckland, New Zealand, 2019, pp. 1819-182. doi:10.1109/ITSC.2019.8917353

Professional Service

Reviewer IEEE Transactions on Vehicular Technology (T-VT)
IEEE Conference on Control Technologies and Applications (CCTA)
American Control Conference (ACC)
ACM Transactions on Intelligent Systems and Technology (TIST)

Abstract

Bilevel Optimization in Learning and Control with Applications to Network Flow Estimation

by

Francesco Seccamonte

The proliferation of complex interconnected systems in today's world has necessitated the development of advanced methods for optimizing their operation and management. Bilevel Optimization (BO), a powerful mathematical framework that considers optimization problems within optimization problems, has emerged as a promising approach to tackle these challenges. This thesis delves into the realm of BO with a primary focus on its application to learning and control in complex systems, particularly addressing the critical problem of network flow estimation.

The core objective of this thesis is to develop novel physics-inspired learning techniques, to provide high-performing and explainable network flow estimators. In Chapter 1, a comprehensive overview of BO is provided, emphasizing its relevance and significance in real-world scenarios. The mathematical foundations of BO problems and the challenges posed by their computational complexity are elucidated, and some numerical schemes to solve them, in exact or approximate form, are reviewed. A collection of some known BO problems in machine learning is presented, and novel connections between BO and problems in machine learning and control are established.

In Chapters 2 and 3 two novel flow estimation algorithms are proposed, addressing different nuances of the flow estimation problem. Both algorithms are rooted in first principles physics, and result into two different Implicit Neural Network Layers. Our approach enables high modularity, and its effectiveness is validated both theoretically

as well as empirically. Extensive experiments across different application domains are presented, namely power systems, water distribution networks and traffic systems.

Finally, Chapter 4 summarizes the findings of this thesis, and highlights potential future research directions.

Contents

Chapter 1

Bilevel Optimization in Learning and Control

This chapter presents an overview of Bilevel Optimization (BO), including numerical methods to solve BO problems. Example BO problems in machine learning and control are presented. Convex optimization is not the main focus of this chapter, but sometimes properties of convex problems are mentioned. The interested reader is referred to [1]. Additionally, a comprehensive review of BO in machine learning can be found at [2].

1.1 An Introduction to Bilevel Optimization

Bilevel Optimization (BO) [3] is a branch of mathematical programming focusing on the interconnection of two nested optimization problems, where the solution to one problem depends on the solution of the other. It stemmed from the seminal work [4] on Stackelberg games, which are examples of BO problems.
Formally, a BO problem can be defined as follows:

$$
\begin{aligned}
z^* = &\arg\min_{z \in \mathcal{Z}} J(z, x^*) && \text{(Outer Cost)} \\
&\text{st.} \quad g(z, x^*) \leq \mathbb{0} && \text{(Outer Constraints)} \\
&\quad x^* = \arg\min_{x \in \mathcal{X}} L(x, z^*) && \text{(Inner Cost)} \\
&\qquad \text{st.} \quad h(x, z^*) \leq \mathbb{0} \ , && \text{(Inner Constraints)}
\end{aligned}
$$

where $z \in \mathcal{Z}$ is the outer (vector-valued) optimization variable, $x \in \mathcal{X}$ is the inner (vector-valued) optimization variable, J and L are the outer and inner cost functions, and g and h are the optional (equality and inequality) outer and inner constraints; the inner problem is itself a constraint for the outer problem. The outer and inner problems and variables are often referred to as upper and lower problems and variables, respectively.

Zucchet and Sacramento [2, Sec. 2] intuitively show how, in general, "[...] the inner optimization process describes what the system does and the outer loss function ultimately measures how good the result of this process is". Depending on the context, the "system" can be a dynamical system responding to an external excitation, a controlled dynamical system, a general optimization problem, a (machine learning) model performing regression or classification, etc. .

1.2 Numerical Methods for Bilevel Optimization

Without any further assumption, BO problems are in general NP-hard to solve [5]. In the following relevant solution approaches are presented, relying either on additional problem assumptions or on numerical approximations to render BO problems tractable.

1.2.1 Approximate Solutions via Gradient Unrolling

The key idea behind [6, 7] is to consider the inner problem as a dynamical system. Loosely speaking, if the state x of an autonomous discrete-time dynamical system is observed for many time steps T, it reaches a fixed point x^* (provided the system has at least a stable one). The evolution of the state towards the fixed point is governed by the dynamics itself, meaning it can be "reversed" in order to compute $\frac{\partial x^*}{\partial x_0}$ via the chain rule. Once the gradient is available, standard gradient-based methods can be applied to the overall BO problem. We now formalize this summary, following the treatment in [7] and the unpublished notes [8].

It is assumed that the inner optimization problem has a unique minimizer for any fixed outer variable value z_j (i.e., the value of z at outer gradient step j), as it is the case in convex problems. The two approximation schemes discussed in the following consist of two distinct phases: a common *forward phase*, corresponding to a forward pass when embedded in a Neural Network; and a *backward phase*, corresponding to the backward pass (i.e., gradient computation) when in a Neural Network context.

Forward Phase In the forward phase, the goal is to solve the inner problem for a given value of the outer variable z_j. By denoting the initial value of the inner optimization variable as x_0, a gradient based method is applied for T steps, resulting in the following intermediate iterates:

$$x_{k+1} = \Gamma(x_k, z_j) \ , k = \{0, \ldots, T-1\} \ .$$

At the end of the forward phase, x_T denotes the approximate solution to the inner problem, which is arbitrarily close to the optimal value x^*.

Backward Phase Once the forward phase is complete, the goal is to compute an expression for $\nabla_z J$, in order to apply gradient based methods to the overall BO problem. By denoting

$$\begin{aligned} B_k &= \frac{\partial \Gamma(x_k, z_j)}{\partial x_k} \ , \ k = \{0, \dots, T-1\} \\ C_k &= \frac{\partial \Gamma(x_k, z_j)}{\partial z_j} \ , \ k = \{0, \dots, T-1\} \ , \end{aligned}$$

an expression for $\nabla_z J$ is given by:

$$\nabla_z J = \frac{\partial J}{\partial x_T} \sum_{k=0}^{T-1} \left(\prod_{s=k}^{T-1} B_s \right) C_k \ . \tag{1.2.1}$$

Depending on how the terms in Eq. (1.2.1) are computed, Forward- or Reverse Mode Differentiation (FMD, RMD) are obtained.

In FMD, an intermediate quantity $Z_{k+1} = \frac{\partial x_{k+1}}{\partial z_j}$ is computed on the fly, resulting in Algorithm 1:

Algorithm 1 FMD, joint forward and backward phases.

```
Require: Current value of $z_j$, $x_0$
Ensure: $\nabla_z J$
1: $Z_0 \leftarrow \mathbb{0}$
2: for k=0 to T-1 do
3:     $x_{k+1} = \Gamma(x_k, z_j)$
4:     $B_{k+1} = \frac{\partial \Gamma(x_{k+1}, z_j)}{\partial x_{k+1}}$
5:     $C_{k+1} = \frac{\partial \Gamma(x_{k+1}, z_j)}{\partial z_j}$
6:     $Z_{k+1} = B_{k+1} Z_k + C_{k+1}$
7: end for
8: return $\nabla_{x_T} J \cdot Z_T$
```

For this reason, FMD as shown in Algorithm 1 is appealing in streaming applications; however, it comes with a high computational cost, due to the matrix multiplication performed at line 6.

On the other hand, to better understand the RMD algorithm it is useful to write the approximate BO problem as follows (dropping the constraints for ease of exposition):

$$z^* = \underset{z\in\mathcal{Z}, x_1,\dots,x_T\in\mathcal{X}}{\arg\min} J(z, x_T) \tag{1.2.2a}$$

$$\text{st. } x_{k+1} = \Gamma(x_k, z),\ k \in \{0,\dots,T-1\} \tag{1.2.2b}$$

The Lagrangian associated to problem (1.2.2) is:

$$\mathcal{L}(x, z, \alpha) = J(z, x_T) - \sum_{k=0}^{T-1} \alpha_k \left(\Gamma(x_k, z) - x_{k+1}\right)\ , \tag{1.2.3}$$

with α_k being the Lagrange multipliers.

Computing the partial derivatives of the Lagrangian (1.2.3) and setting the relative quantities to 0 leads to the same expression as Eq. (1.2.1); however, it is now easier to instantiate the RMD algorithm with distinct forward and backward phases, as in Algorithm 2:

RMD as detailed in Algorithm 2 shows two separate loops for the forward (line 1) and backward (line 8) phases, and compared to FMD has lower computation due to fewer matrix multiplications; however, it is not suitable for streaming applications and it has high memory requirements, due to the need to store the entire inner optimization trajectory (lines 3-4).

Effectiveness of FMD and RMD Researchers in [7] show how the FMD and RMD approximation procedures are effective even when the inner problem does not possess

Algorithm 2 RMD, forward and backward phases.

```
Require: Current value of $z_j$, $x_0$
Ensure: $\nabla_z J$
 1: for k=0 to T-1 do
 2:     $x_{k+1} = \Gamma(x_k, z_j)$
 3:     $B_{k+1} = \frac{\partial \Gamma(x_{k+1}, z_j)}{\partial x_{k+1}}$
 4:     $C_{k+1} = \frac{\partial \Gamma(x_{k+1}, z_j)}{\partial z_j}$
 5: end for
 6: $\alpha_T \leftarrow \nabla_{x_T} J$
 7: $\nabla_z J \leftarrow \mathbb{0}$
 8: for k=T-1 to 0 do
 9:     $\nabla_z J = \nabla_z J + \alpha_{k+1} C_{k+1}$
10:     $\alpha_k = \alpha_{k+1} B_{k+1}$
11: end for
12: return $\nabla_z J$
```

a unique minimizer, and empirically analyze the effect of varying the number of inner iterations T (which is itself a hyperparameter to be tuned manually).

However, recent research [9] shows how approximation methods may introduce bias in the gradient estimate, especially when the inner problem does not possess a unique minimizer.

1.2.2 Solution via Implicit Differentiation

The approximations presented in the previous section rely on the assumption of a differentiable inner problem having a unique minimizer, but have been shown to be quite effective even when that is not the case in practice. When the inner problem possesses the additional property of convexity, an exact solution approach to a BO problem can be derived by using Implicit Differentiation (ID).

The optimizer of a strongly convex and differentiable optimization problem satisfies the system of linear equations given by its Karush–Kuhn–Tucker (KKT) conditions equaling the zero vector [1, Ch. 5]. In the remainder of this section, we follow the derivation in [10, 11] adapting it to our framework. For ease of exposition, we drop the Inner Constraints; a more detailed derivation involving constraints is given in Chapter 2.

The Lagrangian associated to the inner problem of (1.1.1) without constraints is

$$\mathcal{L}_{\text{in}}(x, z) = J(x, z)$$

At the optimum, we have $\nabla_x \mathcal{L}_{\text{in}}(x, z)|_{x^*} = \mathbb{0}$. We are now left with computing the derivative of the inner problem with respect to the parameter z: once available, standard gradient based methods can be applied for solving the overall BO problem.

To this end, we can alternatively view the inner problem in (1.1.1) as a mapping $s : \mathcal{Z} \to \mathbb{X}$ from parameters z to solution x^*; our goal is to compute $\frac{\partial s}{\partial z}$.

Thanks to the implicit function theorem [12, Ch. 9] this quantity is equal to:

$$\frac{\partial s}{\partial z} = -\left(\frac{\partial}{\partial x} \nabla_x \mathcal{L}_{\text{in}}(x, z) \right)^{-1} \frac{\partial}{\partial z} \nabla_x \mathcal{L}_{\text{in}}(x, z) \bigg|_{x=x^*}, \tag{1.2.4}$$

The gradient of the overall BO problem is then given by

$$\nabla_z J = \frac{\partial J}{\partial x}\bigg|_{x=x^*} \frac{\partial s}{\partial z}$$

and it can be used for standard gradient based methods.

The main disadvantage of ID lies in the computation of the matrix inverse in (1.2.4), which renders it intractable for large-scale problems.

1.2.3 Notes on Numerical Solution Schemes

When looking at the solutions schemes to BO problems reviewed in this section, an interesting observation arises when adopting a dynamical system perspective.

It is possible to consider a BO problem as the interconnection of two dynamical systems, corresponding to the outer and inner optimization problems, respectively. Then, when adopting FMD/RMD, we are essentially assuming that the dynamics of the inner problem is much faster than the outer problem, or at least fast enough given a specific value of inner iterations T. When adopting ID, the assumption is pushed even further: The inner problem is considered to have negligible dynamics, as it always operates at its optimum (that is, its KKT conditions are enough to characterize it, and there is no transient). We elaborate more on this topic in section 1.4.

1.3 Bilevel Optimization in Machine Learning

1.3.1 Hyperparameter Optimization

In machine learning, hyperparameters are parameters that are not learned from data but are set prior to the training process. They control various aspects of the learning process, such as the learning rate, the number of hidden layers and units in a neural network, the regularization weight, etc. . Hyperparameter Optimization (HO) [13] involves searching through a predefined space of hyperparameters to find the combination that results in the best performance for a given machine learning task. The goal is to maximize the model's performance metrics, such as accuracy, precision, or F1 score, while avoiding overfitting or underfitting.

Example 1.3.1 (Tikhonov Regularization)**.** Regularized least squares, also known as Ridge Regression or Tikhonov Regularization [14, Chapter 6], is a classical approach to finding the coefficients $x^* \in \mathbb{R}^n$ satisfying the underdetermined system of linear equations $Ax = b,\ A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$. The approach amounts to computing x^* by solving the optimization problem

$$x^* = \arg\min_x \|Ax - b\|_2^2 + \|x\|_{\mathcal{Q}}^2 \ .$$

Even though not explicitly stated, the optimizer x^* is a function of the hyperparameter $\mathcal{Q}$, corresponding to the weight of the regularizer's 2-norm.

Early works employing bilevel optimization techniques for HO date back almost twenty years. [15] and [16] perform HO for Support Vector Machines. The former uses the implicit function theorem under the hood to solve the optimization problem, even though not all of its assumptions are satisfied nor are they stated. On the other hand, the latter employs a carefully designed grid search approach.

More recently, HO formulated as a BO problem has been solved approximately for the case of Kernel Ridge Regression [17] and generic Deep Learning architectures [7]. The former uses the implicit function theorem, whereas the latter applies forward- and reverse-mode differentiation.

Example 1.3.2 (HO for Tikhonov Regularization)**.** Elaborating further on Example 1.3.1, we are interested in optimally choosing the 2-norm regularizer weight $\mathcal{Q}$. The optimality criterion is the cross-validation error $J(\mathcal{Q}, x^*, \bar{x})$, which is computed by holding out some data $\bar{x}$ for validation. The HO problem can be reformulated in BO terms as follows:

$$\begin{aligned} \mathcal{Q}^* &= \arg\min_{\mathcal{Q}} J(\mathcal{Q}, x^*, \bar{x}) \\ &\text{st.} \quad x^* = \arg\min_x \|Ax - b\|_2^2 + \|x\|_{\mathcal{Q}^*}^2 \end{aligned}$$

1.3.2 Meta-learning

Meta-learning [18], often referred to as "learning to learn", is a subfield of machine learning that focuses on designing algorithms and models that can learn and adapt to new tasks or domains more efficiently and effectively. The key idea behind meta-learning is to leverage prior learning experiences to improve the learning process for new, unseen tasks.

From a BO standpoint, the meta-learning objective corresponds to the outer loss function, whereas the inner loss functions amounts to the sum of the task-specific objectives. The inner optimization variables are task-specific parameters to be learned, whereas the outer ones are learnable parameters to be shared across tasks [7].

1.3.3 Model-based Reinforcement Learning

Model-based reinforcement learning (MBRL) [19] is an approach in the field of reinforcement learning (RL) where an agent learns a predictive model of the environment and uses this model to make decisions and plan actions. In contrast to model-free RL, where the agent directly learns a policy or value function from interaction with the environment, MBRL involves constructing a model of how the environment behaves and then using that model for decision-making.

Typically, MBRL is mathematically formulated as:

$$\begin{aligned} &\min_{\pi} J(\pi, x_{1:T}) \\ &\qquad \text{s. t. } x_{t+1} \sim P(x^{+}|x_t, a_t) \\ &\qquad\qquad a_t \sim \pi(a, x_t) \ , \end{aligned}$$

where J is the cost (or a negative reward) to be minimized, π is the policy to be learned,

x is the state of the environment, a_t is the action at time t drawn from policy π, and P the (known) transition dynamics.

Oftentimes, the transition dynamics is modeled as a Markov Decision Process (MDP) [20], to better capture the uncertainty and stochasticity of the environment. However, many dynamical systems tend to evolve towards minimal energy states [21, 22, 23, 24], and the correspondence between certain contracting dynamics and optimization problems has been established [25]. This observation allows us to reformulate the transition dynamics of MBRL as an optimization problem, recovering the standard BO formulation.

1.4 Bilevel Optimization in Systems and Control

1.4.1 Differentiable Model Predictive Control

Model Predictive Control (MPC) [26, 27] is a control strategy widely used in engineering and control systems. It is a mathematical optimization-based approach that repeatedly solves an optimization problem over a finite prediction horizon to determine control inputs for a dynamical system. The primary goal of MPC is to optimize a specified performance criterion while satisfying system constraints. At each time step, MPC uses a model of the system's behavior to predict future states and then calculates control inputs that minimize a cost function, subject to constraints, based on these predictions.

In recent years, there has been an interest for MPC problems where cost, constraints and/or dynamics are only partially known, and parametrized by a certain parameter z. Determining the optimal value of such parameter z requires differentiating through the solution of the MPC problem: this formulation is referred to as Differentiable MPC (Diff-MPC).

Given a dynamical system $x_{k+1} = Ax_k + Bu_k$, Diff-MPC can be instantiated as follows:

$$
\begin{aligned}
& u^*(z) = \arg\min_u J(u, x(z)) && \text{(Cost function)} \\
\text{st.}\quad & x_{k+1} = A(z)x_k + B(z)u_k,\ k = \{0, \dots, N-1\} && \text{(Dynamics)} \\
& (x_k, u_k) \in \mathcal{X}(z) \times \mathcal{U}(z),\ k = \{0, \dots, N-1\} && \text{(Constraints)} \\
& x_N \in \mathcal{X}_N(z) && \text{(Terminal Constraints)}
\end{aligned}
$$

The above formulation can be seen as the inner problem of a BO problem; it is a simplification of the imitation learning frameworks in [28, 29], and of the Diff-MPC based on Neural State-Space Models in [30].

1.4.2 Feedback Optimization

Similarly to Diff-MPC, the recent research area of Feedback Optimization [31] (also known as Online Optimization or Closed-loop Optimization) focuses on the interconnection of an optimization algorithm with a dynamical system, where data used for training comes from the system itself, in an online fashion.
For the two classes of nonlinear systems of the form

$$
\begin{aligned}
\dot{x} &= f(x, u) \\
y &= g(x) + d\ ,
\end{aligned}
$$

and linear time-invariant systems with additional process disturbance, the goal is to

simultaneously compute an optimal input u and trajectory y minimizing a cost function, make the system track such a trajectory, and reject disturbances. Extensions to handle some constraints have recently been proposed [32].
To achieve the above mentioned goals, the system dynamics is generally assumed to be much faster than the process optimizing u and y, with a negligible transient. If the system can be expressed as a gradient flow, the assumption corresponds to solving a BO problem via ID, as shown in Sec. 1.2.

1.4.3 Time-varying Optimization

A slightly different thrust is presented in [25], where the authors focus on time-varying optimization problems, i.e., optimization problems which are parametrized by a quantity that changes over time. By considering the time-varying convex optimization problem as a dynamical system, the goal is to quantify the tracking error of such a system, where the tracking signal is parameter-dependent.
Rather than assuming an extremely fast dynamics, authors focus on bounding the tracking error of the dynamical system in terms of its contraction factor [33].

Their time-varying optimization problem can be seen as the inner problem of a BO scheme, and the analysis can be useful when applying approximate schemes to solve BO problems, such as FMD/RMD.

1.5 Open Software for Bilevel Optimization

Traditionally, the Machine Learning community has been notorious for sharing software and data with permissive open source licenses. Since it has taken interest in BO, a plethora of free software packages have become available.

In recent years, `qpth` [10] has been the first effort towards providing differentiable quadratic program layers in `pytorch`. It leverages ID, and the authors released also an extension tailored to Diff-MPC (`mpc.pytorch`, [28]). Similarly, `cvxpylayers` [34] provides differentiable convex optimization layers by leveraging ID with integration for `pytorch`, `Tensorflow` and `JAX`. Compared to `qpth`, it can handle a broader variety of convex problems. `JAXopt` [35] enables differentiating through fixed point maps (including convex problems) via implicit differentiation, and it targets the `JAX` ecosystem.

`higher` [36] is a Python package specifically designed for meta-learning, and can be used to automatically and efficiently track gradients in `pytorch` via FMD/RMD.

`theseus` [37] is the latest effort from Facebook Research, superseding `higher` and providing tools for Differentiable Nonlinear Least-Squares by leveraging both ID as well as FMD/RMD. Similarly, `TorchOpt` [38] is a differentiable optimization package built on top of `pytorch`, with an emphasis on a functional programming API and including both ID as well as FMD/RMD as BO solution methods.

`NeuroMANCER` [39] is an ongoing effort at Pacific Northwest National Laboratory tailoring generic differentiable programming, including convex problems. As of now, the solutions to BO is obtained by reversing the inner gradient iterations via the built-in `pytorch`'s autograd module.

Chapter 2

Flow Estimation in Infrastructure Networks

The content of this chapter is based on the peer-reviewed publication [40][1].

2.1 Introduction

Many societal engineered systems can be described as a flow network carrying a commodity. Physical infrastructures belong to this class: traffic, power, water, and gas distribution systems all have in common (i) an underlying network structure and (ii) an ability to carry flows of commodities according to some physical law. Oftentimes, only limited sensing is available on these infrastructures so that not all flows are available and accurately measured. In short, the problem of estimating flows of commodities in a network based upon partial and noisy observations is an important task of potential societal value.

[1]©2023 IEEE. Reprinted, with permission, from F. Seccamonte, A. K. Singh, and F. Bullo, Inference of infrastructure network flows via physics-inspired implicit neural networks, in 2023 IEEE Conference on Control Technologies and Applications (CCTA), 2023.

The Control Engineering community has long taken interest in infrastructure networks for several monitoring, estimation and control tasks: load forecasting [41] and admittance estimation [42] in power networks, leak detection [43] and state estimation [44] in water networks, traffic estimation [45] in road networks. Many of the aforementioned works, however, are heavily task-specific, and are of difficult applicability even in case different sensor data is available.

On the other hand, the Deep Learning community has only recently become interested in the flow estimation problem, initially adapting recurrent architectures for this task [46, 47]. More recent approaches belong to the rising physics-cognizant learning field, which aims at integrating domain knowledge in learning algorithms [48, 49, 50].

A first attempt at estimating flows by assuming conservation of mass has been proposed in [51]. Mass conservation is encouraged by solving a regularized divergence minimization problem, however network parameters are not accounted for. [52] extends the divergence minimization problem by learning a regularizer as a function of edge features, and currently achieves state of the art results. In Chapter 3, we look at the flow estimation problem in the presence of unknown network parameters; however, the full injection (also known as supply/demand) vector is assumed to be known. None of the aforementioned approaches is able to handle edge capacities or operational constraints.

This work contributes a framework to jointly learn network parameters and infer missing flows and injections in an infrastructure network. The resulting architecture is highly modular, and consists of two main components: an upstream block learns network parameters from node and edge features, and a downstream block infers missing flows and injections, while accounting for physical laws and enforcing constraints satisfaction. The architecture can be trained in an end-to-end fashion with standard gradient-based methods; at the same time, the two clearly disjoint but synergistic modules greatly lend explainability to the approach. While being physics-inspired, the proposed architecture is application-agnostic, hence it is suitable for different infrastructures.

Specifically, our contributions can be summarized as follows: 1) we establish a connection between flow estimation problems and the Thomson's Principle from the circuits theory literature; 2) we generalize the flow estimation problem to also include injections and account for edge capacities and more general operational constraints; 3) leveraging recent results [10, 11] on implicit differentiation, we propose an end-to-end architecture, composed of a Graph Neural Network and an Implicit Layer, to jointly estimate missing flows, injections as well as network parameters from graph structure, node and edge features; 4) we empirically show how the proposed approach outperforms the state of the art on two different case studies (power and traffic networks).

2.2 Notation and Problem Statement

An n-dimensional column vector of (positive) real numbers is denoted as $\mathbf{x} \in \mathbb{R}^n$ ($\mathbf{x} \in \mathbb{R}^n_{>0}$). An n-dimensional column vector of ones (zeros) is denoted as $\mathbf{1}_n$ ($\mathbf{0}_n$). The symbol $\mathbf{I}_{m\times m}$ denotes the identity matrix of size m. The symbol $\mathbf{0}_{m\times n}$ denotes a matrix of zeros of size $m \times n$. Given two vectors $\mathbf{a}, \mathbf{b}$, we define $(\mathbf{a}, \mathbf{b}) = [\mathbf{a}^T \; \mathbf{b}^T]^T$. The symbol $\mathbb{1}_n^\perp$ denotes the subset of $\mathbb{R}^n$ orthogonal to the vector $\mathbf{1}_n$. The symbol $\|\cdot\|_\mathcal{A}$ denotes the Euclidean 2-norm weighted by $\mathcal{A}$, with $\mathcal{A}$ positive definite. We denote a graph as $\mathcal{G}(\mathcal{V}, \mathcal{E})$, its oriented incidence matrix is $B \in \mathbb{R}^{n\times m}$, its node and edge sets as $\mathcal{V}, |\mathcal{V}| = n$ and $\mathcal{E}, |\mathcal{E}| = m$, respectively.

We are given an infrastructure network represented as a (directed) graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Both node and edge sets are known. The flows and injections of the network are collected in the vectors $\mathbf{f} \in \mathbb{R}^m$, $\mathbf{p} \in \mathbb{R}^n$. When in operation, some sensors allow to measure $\mathbf{p}' \in \mathbb{R}^{n'}$ injections and $\mathbf{f}' \in \mathbb{R}^{m'}$ flows, $n' \le n$, $m' \le m$. The goal is to infer missing flows $\hat{\mathbf{f}} \in \mathbb{R}^{m-m'}$ and injections $\hat{\mathbf{p}} \in \mathbb{R}^{n-n'}$.

2.3 Physics of Infrastructure Networks

Generally speaking, infrastructure networks are often characterized by both flow ($\mathbf{f} \in \mathbb{R}^m$) and effort ($\mathbf{x} \in \mathbb{R}^n$) variables, which are usually linked through a constitutive relation of the form $\mathbf{f} = \mathcal{A}h(B^T\mathbf{x}), h : \mathbb{R}^m \to \mathbb{R}^m$.

The function h is usually monotone strictly increasing, with $h(0) = 0$. Linearizing the map h around the origin, the relations in a lossless network read as follows:

$$\mathbf{p} = B\mathcal{A}B^T\mathbf{x} \; , \qquad \mathbf{f} = \mathcal{A}B^T\mathbf{x} \; , \qquad (1a, 1b)$$

$$B\mathbf{f} = \mathbf{p} \; , \qquad \mathbf{p} \in \mathbb{1}_n^\perp \; , \qquad (1c, 1d)$$

with $\mathcal{A} = \text{diag}(a), a \in \mathbb{R}^m_{>0}$. In power networks, $\mathcal{A}$ is the line admittance matrix [53, Sec. 7.4]; for consistency, we refer to it as admittance matrix for other networks as well.

Equations (1c, 1d) denote flow conservation, and in [52, 51] are collectively referred to as the divergence-free condition, assuming $\mathbf{p} = \mathbf{0}_n$. Note that even networks without a constitutive relation, such as traffic [54], satisfy (1c, 1d).

2.3.1 The Thomson's Principle for Linear Laplacian Flows

We recall a classical result about the exact solution to linear Laplacian flow networks, as originally stated for DC power networks [55, Sec. 1.3.5].

Theorem 2.3.1 (Thomson's principle and Laplacian flows)**.** *Let* $\mathbf{p} \in \mathbb{1}_n^\perp$ *be a given injection vector. Then the following statements are equivalent:*

(i) $\mathbf{f}^*$ *satisfies the DC-power flow equations (Kirchoff and Ohm), i.e.,* $\exists\ \mathbf{f} \in \mathbb{1}_m^\perp$ *such*

that

$$\mathbf{p} = B\mathcal{A}B^T\mathbf{x}, \quad \mathbf{f} = \mathcal{A}B^T\mathbf{x}\,. \tag{2.3.2}$$

(ii) $\mathbf{f}^*$ *is the unique solution to the optimization problem*

$$\min_{\mathbf{f}} \; \|\mathbf{f}\|^2_{\mathcal{A}^{-1}}, \quad \text{s.t. } \mathbf{p} = B\mathbf{f}. \tag{2.3.3}$$

Intuitively, Theorem 2.3.1 shows how power flows through the path of minimum effective resistance. Such flows can be recovered via the convex optimization problem (2.3.3): While effort variables are latent, the availability of the admittance matrix is essential.

2.3.2 Operational Constraints in an Infrastructure Network

Beside the fundamental equations (2.3.1), operational constraints are often to be considered as well. For instance, in some applications such as traffic networks, flows are non-negative. Additionally, capacity and/or lower bounds could be available via system specifications or inferred thanks to expert knowledge. Such constraints can be collected into the linear inequality $\mathbf{c} \leq G(\mathbf{f}, \mathbf{p})$, with $G \in \mathbb{R}^{d\times(m+n)}$.

2.4 End-to-end Optimal Flow Inference

In the previous section, we showed how inferring missing flows while enforcing flow conservation via problem (2.3.3) requires knowledge of the admittance matrix $\mathcal{A}$. However, often times admittances are not readily available; other times, they are a (possibly empirical or unknown) function of node and edge features $\mathcal{N}^f, \mathcal{E}^f$.
We therefore propose an architecture to model the function $(\hat{\mathbf{f}}, \hat{\mathbf{p}}) = F(\mathcal{G}, \mathcal{N}^f, \mathcal{E}^f, \mathbf{f}', \mathbf{p}')$,

accompanied by an algorithm to efficiently learn such function in an end-to-end fashion from limited data.

2.4.1 Jointly Learning Admittances and Network Flows

Optimizing problem (2.3.3) over admittances as well as missing flows and injections would result in a nonconvex problem. Such an approach trivially pushes the admittance vector to the zero vector, and disregards node and edge features. For this reason, we define a soft constrained version of problem (2.3.3) with the additional linear inequalities as (2.4.1b) and embed it within another optimization problem:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^m_{>0}} \mathbb{E}\left[\|\mathbf{z}^* - \hat{\mathbf{z}}\|^2\right] =: \Psi(\mathbf{z}^*(\boldsymbol{\theta})) \tag{2.4.1a}$$

$$\begin{aligned} \text{s.t. } \mathbf{z}^* = \arg\min_{\mathbf{z}} \| \begin{bmatrix} \mathbf{I}_{m\times m} & \mathbf{0}_{m\times n} \end{bmatrix} \mathbf{z}\|^2_{\mathcal{A}(\boldsymbol{\theta})^{-1}} + \|D\mathbf{z}\|^2_2 \\ \text{s.t. } \mathbf{c} \le G\mathbf{z}\ , \end{aligned} \tag{2.4.1b}$$

where for ease of notation we define the state $\mathbf{z} := (\mathbf{f}, \mathbf{p})$, $D = \begin{bmatrix} B & -\mathbf{I}_{n\times n} \end{bmatrix}$ and the admittance matrix $\mathcal{A}$ is parametrized by $\boldsymbol{\theta}$. The cost function (2.4.1a) represents the mismatch between the inferred state $\mathbf{z}^*$ and (a subset of) missing states $\hat{\mathbf{z}}$, and problem (2.4.1b) optimizes on the missing flows and efforts only. Imposing flow conservation as a soft constraint is beneficial when dealing with noisy observations.

Problem (2.4.1) is a bilevel optimization problem [3]: solving the outer problem (2.4.1a) requires knowledge of the solution of the inner problem (2.4.1b), which in turn depends on the outer problem solution. Bilevel optimization problems naturally arise in meta-learning and hyperparameter optimization [7]. They are NP hard in general, however gradient based approaches have been developed to compute approximate solutions

[7]. In the following, we adopt a different approach based on implicit differentiation (ID), enabling an efficient solution of problem (2.4.1) to a local minimum.

Remark 2.4.1 (More on the Outer Problem)**.** The objective function of the outer problem (2.4.1a) represents a cross-validation error, and can be obtained as follows: Some measured flows and injections are randomly held out, and treated as states to be estimated in the inner problem (2.4.1b). This approach allows the generation of multiple inner problem instances, even when data is extremely limited.

Remark 2.4.2 (Extension to efforts inference)**.** Despite being mainly interested in the flows and injections inference, it is possible to extend the problem formulation (2.4.1) to include also effort variables. The variable $\mathbf{z}$ can be defined as $\mathbf{z} := (\mathbf{f}, \mathbf{p}, \mathbf{x})$, the zero matrix in problem (2.4.1b) can be modified to $\mathbf{0}_{m\times 2n}$ and the relations between efforts and flows and injections, as specified in Eq. (2.3.1), can be accounted for by suitably modifying the matrix D in problem (2.4.1b). Similarly, constraints on efforts (e.g., nodal pressures in water networks) can be accounted for by modifying the inequality $\mathbf{c} \leq G\mathbf{z}$ in (2.4.1b).

2.4.2 Leveraging Convexity of the Inner Problem

The inner problem (2.4.1b) is strongly convex and differentiable. Therefore we can replace it with a system of linear equations, corresponding to its Karush–Kuhn–Tucker (KKT) conditions equaling the zero vector. In the remainder of this section, we adapt the derivation in [10, 11] to our framework.

The Lagrangian associated to the inner problem (2.4.1b) is

$$\mathcal{L}(\mathbf{z}, \boldsymbol{\lambda}, \boldsymbol{\theta}) = J(\mathbf{z}, \boldsymbol{\theta}) - \boldsymbol{\lambda}^T(G\mathbf{z} - \mathbf{c}),$$

with $J(\mathbf{z}, \boldsymbol{\theta})$ being the cost function of (2.4.1b) and $\boldsymbol{\lambda} \in \mathbb{R}^d_{\geq 0}$ the Lagrange multiplier associated to the inequality constraints. For such problem, the KKT conditions are necessary and sufficient for optimality; they can be compactly rewritten as

$$g(\mathbf{z}, \boldsymbol{\lambda}, \boldsymbol{\theta}) := \begin{bmatrix} \nabla_{\mathbf{z}} \mathcal{L}(\mathbf{z}, \boldsymbol{\lambda}, \boldsymbol{\theta}) \\ -\mathrm{diag}(\boldsymbol{\lambda})(G\mathbf{z} - \mathbf{c}) \end{bmatrix}, \tag{2.4.2}$$

and at the optimum we have $g(\mathbf{z}^*, \boldsymbol{\lambda}^*, \boldsymbol{\theta}) = \mathbf{0}_{m+n+d}$.

In order to make the inner problem amenable to standard training via gradient-based methods, typically used in machine learning architectures, the derivative of the inner problem with respect to the parameter $\boldsymbol{\theta}$ needs to be computed.

To this end, we can alternatively view the problem (2.4.1b) as a mapping $s : \mathbb{R}^m \to \mathbb{R}^{(m+n)}$ from parameters $\boldsymbol{\theta}$ to solution $\mathbf{z}^*$. Let $\boldsymbol{\kappa} := (\mathbf{z}, \boldsymbol{\lambda})$; our goal is to compute $\frac{\partial s}{\partial \boldsymbol{\theta}}$. Thanks to the implicit function theorem this quantity is equal to:

$$\frac{\partial s}{\partial \boldsymbol{\theta}} = -\left(\frac{\partial g(\boldsymbol{\kappa}, \boldsymbol{\theta})}{\partial \boldsymbol{\kappa}} \right)^{-1} \frac{\partial g(\boldsymbol{\kappa}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \Bigg|_{\boldsymbol{\kappa} = \boldsymbol{\kappa}^*}, \tag{2.4.3}$$

with

$$\frac{\partial g(\boldsymbol{\kappa}, \boldsymbol{\theta})}{\partial \boldsymbol{\kappa}} = \begin{bmatrix} \frac{\partial}{\partial \mathbf{z}} \nabla_{\mathbf{z}} \mathcal{L}(\mathbf{z}, \boldsymbol{\lambda}, \boldsymbol{\theta}) & -G^T \\ -\mathrm{diag}(\boldsymbol{\lambda}) G & -\mathrm{diag}(G\mathbf{z} - \mathbf{c}) \end{bmatrix},$$
$$\frac{\partial g(\boldsymbol{\kappa}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial}{\partial \boldsymbol{\theta}} \nabla_{\mathbf{z}} \mathcal{L}(\mathbf{z}, \boldsymbol{\lambda}, \boldsymbol{\theta}) \\ \mathbf{0}_{d \times (m+n)} \end{bmatrix}.$$

In summary, the inner problem (2.4.1b) has been cast to an implicit layer (IL) [56], which can be easily embedded in a variety of learning architectures.

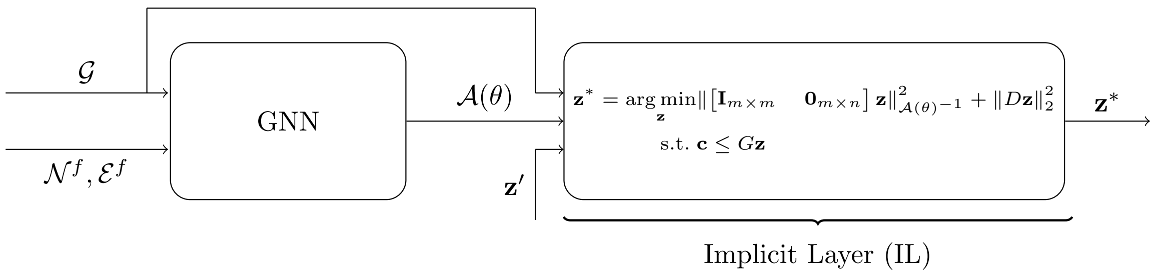
$\mathcal{G}$
$\mathcal{N}^f, \mathcal{E}^f$
GNN
$\mathcal{A}(\theta)$
$\mathbf{z}'$
$\mathbf{z}^* = \arg\min_{\mathbf{z}} \| [\mathbf{I}_{m\times m} \quad \mathbf{0}_{m\times n}] \, \mathbf{z}\|^2_{\mathcal{A}(\theta)^{-1}} + \|D\mathbf{z}\|_2^2$
s.t. $\mathbf{c} \leq G\mathbf{z}$
$\mathbf{z}^*$
Implicit Layer (IL)

Figure 2.1: Schematics of the proposed architecture. A graph representation of the infrastructure $\mathcal{G}$ is given as input to a Graph Neural Network GNN, together with node and edge features $\mathcal{N}^f, \mathcal{E}^f$. The GNN outputs admittances $\mathcal{A}(\boldsymbol{\theta})$, which are used as inputs to the implicit layer IL. Additionally, IL takes the graph $\mathcal{G}$ and the measured (i.e., known) states $\mathbf{z}'$ as inputs, and produces the estimated states $\mathbf{z}^*$ as outputs. During training, IL outputs the estimated states $\mathbf{z}^*$ for multiple problem instances, which can then be employed to compute a cross-validation error as loss criterion.

2.4.3 Integrating the IL in a Deep Learning pipeline

The Implicit Layer developed in the previous subsection models problem (2.4.1b), and allows computation of gradients with respect to the parameter $\boldsymbol{\theta}$.
The task of modeling a learnable function to estimate admittances can now be addressed. We employ a graph neural network (GNN), motivated by its expressivity, its ability to process node and edge features as well as graph structures in an explainable way, and its modularity and the possibility of training it via gradient descent.

GNN on Graph vs. Line Graph Beside the choice of the specific GNN layers, our architecture differs from the one in [52] for the representation of the infrastructure network in the GNN. In [52], the GNN acts on the line graph $\mathcal{L}(\mathcal{G})$ [57] associated to the physical network, neglecting node features. Only for some networks, e.g. traffic networks, representing links as nodes is a commonly accepted choice [54].

However, a GNN acting on the line graph is potentially unable to distinguish two very different graphs, as highlighted in the following lemma:

Lemma 2.4.3. *The map $\mathcal{G} \mapsto \mathcal{L}(\mathcal{G})$ is not injective.*

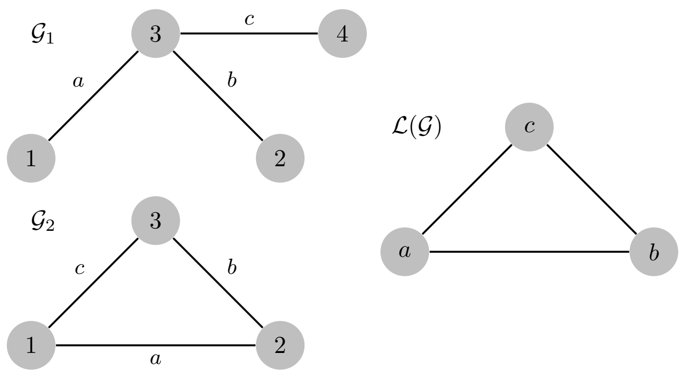

Figure 2.2: Proof of Lemma 2.4.3 by counterexample. Two distinct graphs $\mathcal{G}_1$, $\mathcal{G}_2$ (left) associated to the same line graph $\mathcal{L}(\mathcal{G})$ (right). Notice how connectivity of the infrastructure network - the only graph property we can assume - is not sufficient for the map $\mathcal{G} \mapsto \mathcal{L}(\mathcal{G})$ to be injective.

Proof: Proof by counterexample. See Fig. 2.2. ■

Lemma 2.4.3 motivates the choice of employing a GNN acting on the original graph. Specifically, we adopt a GNN with 2 graph attention (GAT) layers accounting also for edge features [58] and `relu` activations. Since the network needs to output line admittances (that is, an edge-related quantity), on top of the GAT layers we add a Multi-Layer Perceptron consisting of one linear layer and `relu` activation applied edge-wise. The GNN outputs the vector $\boldsymbol{\theta} \in \mathbb{R}^m_{>0}$ which parametrizes the admittance matrix $\mathcal{A}(\boldsymbol{\theta}) = \text{diag}(\boldsymbol{\theta})$. The end-to-end architecture is depicted in Fig. 2.1.

Remark 2.4.4 (Comparison with RMD [52])**.** Albeit based on different theoretical grounds, the proposed architecture shares similarities with the state of the art [52], which relies on approximate solutions to the inner problem via Reverse-Mode Differentiation (RMD) [7]. The main differences with the RMD-based approach are as follows. First, ID allows us to compute an exact derivative (instead of approximate), and to avoid unrolling the inner gradient iterations to apply the chain rule for gradient computation. Second, when nonnegativity constraints are added, there is no need to perform projections after each inner gradient step in ID. Third, more general linear inequalities can be included in ID. Finally, there are two fewer hyperparameters to select in ID - the number of inner iterations to be performed and the learning rate of the inner problem. The computation of the matrix inverse in (2.4.3) can be efficiently handled via a LU decomposition, as implemented in `qpth` [10].

Remark 2.4.5 (Modularity of the proposed approach)**.** As shown in Fig. 2.1, there is a clear separation between admittances computation (GNN), and physics-based states estimates (IL). This modularity in principle allows the GNN block to be replaced with any other module whose outputs are generalized admittances. Moreover, employing IL instead of RMD as in [52] lowers the interplay between the two blocks during training: ID loosens the interdependence between hyperparameters of the two blocks, since the number of iterations performed in IL does not affect the choice of learning rate and/or number of iterations for the GNN. Such modularity is beneficial in terms of ease and speed of hyperparameter tuning, and empowers the user to effortlessly try different admittances estimators. Finally, the output of the GNN block can easily be inspected to ensure admittances obey prior/expert knowledge, benefiting the explainability of the approach.

2.4.4 Lowering the Complexity of the Inner Problem via Divide and Conquer

Computing a forward pass of IL requires solving a constrained convex problem. The backward pass requires computing the matrix inverse in Eq. (2.4.3). algorithms to perform such tasks scale roughly cubically in the number of optimization variables.

To reduce the computational complexity of IL, we propose a divide and conquer approach, consisting of exploiting the available measurements to split the problem into multiple subproblems. Given a weakly connected digraph describing the infrastructure network, we remove all directed edges with measured flows, obtaining a spanning subgraph [59]. We then decompose the spanning subgraph into its weakly connected components and define an appropriate decoupled divergence minimization problem (2.4.1b)

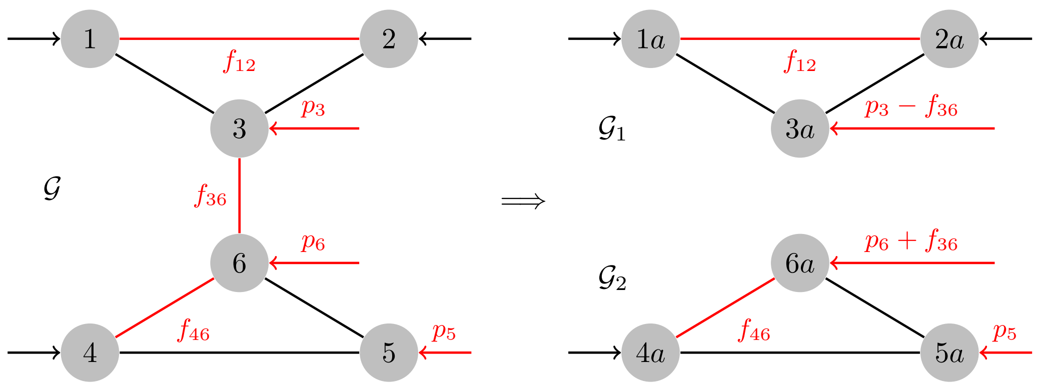

Figure 2.3: Divide and conquer: Splitting a graph $\mathcal{G}$ (left) into two weakly connected components $\mathcal{G}_1, \mathcal{G}_2$ (right). Injections are denoted with arrows, edges connecting two nodes with simple links. Red and black denotes measured and unknown flows/injections, respectively.

for each of them. These problems are based upon absorbing the removed measured flows into the injections vector, as illustrated in Fig. 2.3.

2.5 Experiments

In this section we perform several experiments to assess the validity of the proposed approach. We use the publicly available power and traffic data from [52] in a transductive setting (i.e., training and inference on the same network).

The traffic network roughly corresponds to the Los Angeles county area, and data was collected by the California Department of Transportation; edge features used in traffic are latitude and longitude coordinates, number of lanes, maximum speed, and highway type (motorway, motorway link, trunk, etc.). The power network corresponds to the PyPSA-Eur model in PyPSA [60]; edge features used in power are resistance, reactance, length, and number of parallel lines, nominal power. As node feature, we use the node degree.

We compare our algorithm to the Minimum Divergence (Min-Div) approach [51]

Table 2.1: Flow estimation results for power and traffic datasets. For Bilevel approaches, RMSE is followed by its standard deviation in parentheses. Bold denotes best performance.

	Power		Traffic	
Method	**RMSE (SD)**	**CORR**	**RMSE (SD)**	**CORR**
Min-Div [51]	0.034	0.93	0.071	0.76
MLP	0.069	0.61	0.083	0.00
GCN	0.064	0.64	0.066	0.00
Bil-GCN [52]	0.027 (0.009)	0.94	0.062 (0.008)	0.79
Bil-GCN-IMP	0.026 (0.008)	0.96	0.060 (0.008)	0.85
Bil-GAT-IMP	0.025 (0.008)	0.96	0.062 (0.004)	0.82
Bil-GAT-IMP-C	**0.023 (0.007)**	**0.97**	**0.059 (0.004)**	**0.87**

and to the bilevel optimization with Graph Convolutional Network and Reverse Mode Differentiation (Bil-GCN) [52]. Similarly to [52], we add two physics-unaware baselines: a Multi-Layer Perceptron (MLP) and a Graph Convolutional network (GCN). MLP and GCN attempt to predict flows based on edge features, but ignore flow conservation.

Training the end-to-end architecture is performed by using the Adam optimizer with a learning rate of 10^{-2} selected via grid search. 10-fold cross-validation is employed for each experiment, and the results presented are averaged on the 10 folds. Results are discussed in the relevant subsections.All the code and data is available at: https://github.com/francescoseccamonte/bilevelflow .

2.5.1 Results: Accuracy

We start by presenting the achieved accuracy in Table 2.1. We run Bil-GCN as in [52], a bilevel optimization architecture with two Graph Convolutional layers (8 hidden neurons) and RMD for the inner problem, and compare it with a similar architecture Bil-GCN-IMP. Practically speaking, we replace the flow estimation layer of Bil-GCN with the IL designed in Sec. 2.4, and leave the rest of the architecture unaltered. In both

Bil-GCN and Bil-GCN-IMP we use the GCN designed in [52], which works by computing the line graph [57] associated to the infrastructure network, and using it in place of the original graph. Only edge features are considered. We employ the hyperparameters chosen in [52]. Notably, in Bil-GCN the number of inner iterations is the same for both experiments. We additionally test the proposed architecture (Bilevel Optimization based on GAT layers and Implicit Differentiation), with only 4 neurons in the hidden layer, when constraints $\mathbf{c} \leq G\mathbf{z}$ are included (Bil-GAT-IMP) or not (Bil-GAT-IMP-C). The proposed architecture acts on the original graph and has 30% fewer trainable parameters than Bil-GCN, despite accounting for both node and edge features. The metrics included are Root Mean Square Error (RMSE) and Pearson Correlation (CORR).

As expected, we can observe how Bil-GCN-IMP outperforms Bil-GCN, since Bil-GCN-IMP always computes an exact solution to the inner problem. The two physics-unaware baselines MLP and GCN are unable to accurately predict flows. Additionally, Table 2.1 shows how the bilevel architecture based on GAT achieves the highest accuracy, and how the inclusion of constraints proves to be beneficial.

2.5.2 Results: Complexity of RMD vs. ID

We now analyze the training times of the architectures employed. The experiments are performed in a Colab notebook on a Tesla T4 with 16 GB of memory, and Table 2.2 summarizes the results. Times showed are obtained by timing the entire training loop with 10-fold cross validation, which includes the update of the trainable parameters of the neural network and the execution of auxiliary functions in addition to the forward and backward passes of the inner problem.

For RMD, the same number of inner iterations is performed in both problems, despite the different inner problem sizes ($|\mathcal{E}| \approx 10^3$ for power, $|\mathcal{E}| \approx 10^4$ for traffic). This

Table 2.2: End-to-end training time for RMD vs. IMP. RMD performs a fixed number of inner iterations, whereas IMP always solves the inner problem to numerical optimality.

Method	Training time power $\lvert\mathcal{E}\rvert \approx 10^3$	Training time traffic $\lvert\mathcal{E}\rvert \approx 10^4$
Bil-GCN [52]	244 [s]	**84 [min]**
Bil-GCN-IMP	77 [s]	104 [min]
Bil-GAT-IMP	**72 [s]**	102 [min]
Bil-GAT-IMP-C	86 [s]	308 [min]

hyperparameter choice results in RMD approximating the optimal solution in the smaller problem with high accuracy. A coarser approximation of the solution of the inner problem via RMD results in a lower computational burden (see, e.g., traffic), but it might lower the accuracy of the final flow estimate. Tuning the hyperparameters of the inner problem to compute a sufficiently good approximation is a downside of RMD, that is eliminated altogether in the proposed approach.

As the problem size gets larger (as in traffic), ID becomes computationally more intense, mainly due to the computation of the matrix inverse in Eq. (2.4.3). However, as highlighted in Table 2.1, we note the accuracy given by ID is consistently higher than RMD. Lowering the complexity of ID with techniques similar to [61] is subject of ongoing research.

When dealing with large inner problems as in traffic, empirical evidence showed that the choice of the GNN architecture (GCN, ChebCN, GAT) has little impact on the training time, as solving multiple instances of the inner problem constitutes the main computational bottleneck.

Bil-GAT-IMP-C (the most computationally expensive approach proposed) achieved inference times < 100 ms on power, and < 10 s on traffic.

2.5.3 GAT on Graph vs. GCN on Line Graph

Interestingly, in Table 2.1 we note how the Bil-GAT-IMP architecture, despite having 30% fewer trainable parameters than Bil-GCN, achieves almost identical performance on power, and does only slightly worse on traffic. We believe the high expressivity of GAT allows using a smaller architecture.

2.5.4 Embedding Prior Knowledge in Linear Inequalities

From Table 2.1 we observe how including operational constraints improves performance of the proposed architecture. The constraints are of the form $f_a + f_b \geq l_{ab}$, with l_{ab} being a lower bound (non tight). We randomly sample approximately $0.2 \times (m - m')$ constraints. However, Table 2.2 shows how this comes at a high computational cost during training, especially in large networks. A possible trade-off would be selecting only the d most informative ones. Defining which constraints are informative is subject of ongoing research.

Chapter 3

Flow Networks with Nonlinear Constitutive Relationships

The content of this chapter is based on the peer-reviewed publication [62].

3.1 Introduction

Similarly to Chapter 2, we are interested in the problem of infrastructure networks flow estimation. Specifically, our aim is to incorporate physical knowledge in the flow estimator, and the starting point is once again conservation of flows.

But the conservation law alone is not enough to uniquely determine flow, which is why [51], [52] and [40] rely on heuristic regularizers to select the "best" conservation-respecting flow. In fact, physical networks are often governed by a *pair* of physical laws: the conservation law, and a *constitutive relationship*, which specifies the magnitude and direction of each edge flow based on "effort" variables at each incident node (e.g., pressure or voltage). For example, in DC circuits, currents are conserved according to Kirchoff's current law, and Ohm's law is the constitutive relationship that relates current flows to nodal potentials. The conservation law and the constitutive relationship together define the unique edge flows (and nodal efforts).

3.1.1 Contributions

This paper proposes a model for network flows that embeds both the conservation law and existence of a constitutive relationship. Our model, which we call an *Implicit Flow Network* (IFN), predicts each edge flow using a trainable nonlinear function of latent nodal variables. These latent variables are constrained to a manifold wherein the conservation law is satisfied. In addition to introducing IFN, we offer the following contributions: (i) a contraction algorithm that is able to both evaluate the IFN layer and backpropagate gradients through it, (ii) an explicit upper bound on the number of iterations required by this algorithm, (iii) a rigorous theoretical comparison between IFN and the state-of-the-art flow estimation methods in [51, 52], and (iv) numerical experiments from several AC power networks and water distribution systems that indicate IFN can significantly outperform these baselines on the flow estimation task. Additionally, because IFN requires a nonlinearity with a constrained slope, we provide (v) a novel "derivative-constrained perceptron", which is essentially a trainable activation function with upper and lower bounds on its slope.

3.1.2 Related Work

Network Flow Estimation Flows on graphs are a classical topic in computer science [63], and flow forecasting has long been studied in specific domains like traffic [64], but interest in the flow estimation task from a machine learning perspective appears to be relatively recent. Deep learning algorithms have been used to predict traffic flows [47, 65] and power flows [66], but [51] and [52] appear to be the first papers to propose methods for domain-agnostic flow prediction, based on the notion of divergence minimization.

Implicit Neural Networks IFN belongs to a growing class of models called implicit neural networks, which do not explicitly state the output of the model; rather, they describe a desired relationship between the model's inputs and outputs. In the prevailing implicit framework, the output is defined as a fixed point of a trainable perceptron. This approach was introduced in [67] as a "deep equilibrium network". Subsequent work has developed new frameworks for ensuring the existence of the fixed point and computing it [68, 69, 70, 71, 72]. Other types of implicit neural networks include neural ODEs [73] and layers that solve convex optimization problems [34] and Nash equilibria [74].

Graph Neural Networks Graph neural networks (GNN) are a diverse family of models for network-related learning tasks that incorporate graph structure directly into the model. GNNs can typically be classified into three types, in increasing order of generality [75, §5.3]: convolutional models [76, 77], attentional models [78], and message-passing models [79, 80]. Recently, [81] proposed an implicit graph convolutional network. Analogously, IFN can be interpreted as an implicit message-passing GNN, with flows serving as messages and latent nodal variables acting as an embedding.

3.1.3 Preliminaries and Notation

We follow the same notation adopted in Chapter 2. Additionally, given a directed graph $G = (\mathcal{V}, \mathcal{E})$, the signed incidence matrix $B \in \{-1, 0, 1\}^{|\mathcal{V}| \times |\mathcal{E}|}$ is the matrix with entries

$$B_{i,e} = \begin{cases} 1, & i \text{ is the head of } e \\ -1, & i \text{ is the tail of } e \\ 0, & \text{else} \end{cases}, \quad \forall i \in \mathcal{V} \text{ and } e \in \mathcal{E}$$

Flow	Nodal Variable	$h(y) =$
DC Current	Voltage	y
DC Power	Voltage	y^2
AC Power (lossless)	Voltage Angle	$\sin(y)$
Water Flow Rate	Hydraulic Head	$\mathrm{sign}(y)\|y\|^{0.54}$
Mechanical Force Networks	Position	y

Table 3.1: Examples of physical flow networks and their constitutive relationships.

For an undirected graph, the signed incidence matrix is obtained by assigning an aribtrary orientation to each edge. For each $i \in \mathcal{V}$, let $\mathcal{N}_{\text{in}}(i), \mathcal{N}_{\text{out}}(i) \subset \mathcal{V}$ be the in-neighbors and out-neighbors of i.

Given a vector $x \in \mathbb{R}^n$, we use the notation $[x]$ to denote the diagonal matrix $\mathrm{diag}(x) \in \mathbb{R}^{n\times n}$. Where such notation would be unclear (e.g., may be confused with brackets to indicate order of operations), we fall back on the $\mathrm{diag}(\cdot)$ notation. We write $x^\perp$ to refer to the vector space that is orthogonal to x, i.e., the space $\{x' \in \mathbb{R}^n : x^\mathsf{T} x' = 0\}$. Given a positive definite diagonal matrix $D \in \mathbb{R}^{n\times n}$, we write $||x||_{2,D}$ to represent the weighted 2-norm $||D^{\frac{1}{2}}x||_2$. Given any matrix M, M_i is the ith column vector of M, and $M^{(j)}$ is the transpose of the jth row vector.

3.2 Implicit Flow Networks

IFN is inspired by the physics of network systems. In many physical networks, nodes "communicate" through the exchange of a commodity, like power, water, or force, which can be represented as edge flows. Flows obey a conservation law: for all $i \in \mathcal{V}$,

$$0 = \overbrace{u_i + \sum_{j\in\mathcal{N}_{\text{in}}(i)} f_{(i,j)}}^{\text{net inflow}} - \overbrace{\sum_{j'\in\mathcal{N}_{\text{out}}(i)} f_{(i,j')}}^{\text{net outflow}}, \tag{3.2.1}$$

where $u \in \mathbb{R}^{|\mathcal{V}|}$ are nodal inflows from outside the network, and $f \in \mathbb{R}^{|\mathcal{E}|}$ are the edge flows. Furthermore, the flows are related to nodal variables through a constitutive relationship (CR); there is some strictly increasing function h such that, for all $(i,j) \in \mathcal{E}$,

$$f_{(i,j)} = a_{(i,j)} h(x_i - x_j), \tag{3.2.2}$$

where $a \in \mathbb{R}^{|\mathcal{E}|}$ are edge weights and $x \in \mathbb{R}^{|\mathcal{V}|}$ are nodal "efforts" or "potentials." For example, in DC power networks, the CR is Ohm's law $f_{(i,j)} = r_{(i,j)}^{-1}(x_i - x_j)$, where r are resistances and x are voltages. In lossless AC networks, the CR is the active power flow equation $f_{(i,j)} = a_{(i,j)} \sin(x_i - x_j)$, where the edge weights are a function of line parameters and x are voltage angles [82, §6.4]. In water distribution systems, the CR is the Hazen-Williams formula [83, Sec. 8.15]. Table 3.1 lists several flow networks, the physical interpretation of the effort variables x, and the flow function h.

We propose IFN as a layer that predicts edge flows based on these two physical laws—conservation and the existence of a CR:

Definition 3.2.1 (Implicit Flow Network)**.** An *implicit flow network* (IFN) is a module with the following components:

(i) fixed parameters $0 < d_{\min} \leq d_{\max}$,

(ii) trainable parameters $\theta \in \mathbb{R}^r$ for some r, and

(iii) a family of differentiable functions $h_\theta : \mathbb{R} \to \mathbb{R}$ such that $d_{\min} \leq h'_\theta(y) \leq d_{\max}$ for all $y \in \mathbb{R}$ and $\theta \in \mathbb{R}^r$, which we call *flow functions*.

The module requires each of the following inputs:

(i) a weighted, connected, undirected graph $G = (\mathcal{V}, \mathcal{E}, a)$ with edge weights $a \in \mathbb{R}^{|\mathcal{E}|}_{>0}$, and

(ii) a supply / demand vector $u \in \mathbb{R}^{|\mathcal{V}|}$ such that $\sum_{i \in \mathcal{V}} u_i = 0$.

The module outputs the unique vector $f \in \mathbb{R}^{|\mathcal{E}|}$ for which there exists $x \in \mathbb{R}^{|\mathcal{V}|}$ such that

$$Bf = u \tag{3.2.3}$$

$$f = [a]h_\theta(B^\mathsf{T} x) \tag{3.2.4}$$

where $B \in \{-1, 0, 1\}^{|\mathcal{V}| \times |\mathcal{E}|}$ is the signed incidence matrix of G, and h_θ is applied element-wise. We use the notation $\mathrm{FN}_{h,\theta}(G, u)$ to represent the solution f given inputs G and u, flow functions h, and parameters θ.

We will prove that IFNs are well-posed in Theorem 3.2.2. Note that (3.2.3) and (3.2.4) are just vectorized statements of the conservation law (3.2.1) and the CR (3.2.2), so these two physical laws directly define the output. The IFN's only trainable component is its flow function, parameterized by θ. In practice, we will only make calls to the *inverse* of the flow function when evaluating and backpropagating through IFN layers, so it is convenient to learn the inverse flow function directly.

We emphasize that IFNs are layers that can be situated in more complex architectures, with other models upstream estimating the supply / demand vector, edge weights, or even the topology. For example, in power systems, demand forecasting is a very well-studied problem [41, 84], and one can solve the economic dispatch problem to forecast power generation at each node [85], collectively leading to an estimate of the supply / demand vector.

3.2.1 Evaluating the Implicit Flow Network

Our approach to evaluating the implicit flow network is adapted from [86] and is illustrated in Figure 3.1. Any undirected graph G induces a direct decomposition of

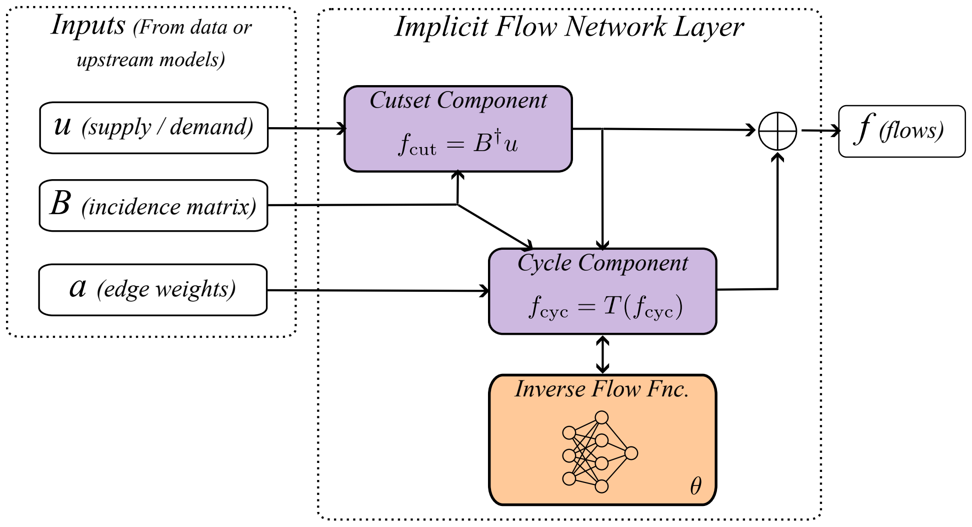
Inputs (From data or upstream models)
u (supply / demand)
B (incidence matrix)
a (edge weights)
Implicit Flow Network Layer
Cutset Component
$f_{\text{cut}} = B^{\dagger}u$
Cycle Component
$f_{\text{cyc}} = T(f_{\text{cyc}})$
Inverse Flow Fnc.
θ
f (flows)

Figure 3.1: Diagram of the IFN. Inputs are the supply / demand vector u, incidence matrix B, and edges weights a, which are either known or output from upstream models. The IFN layer separately computes the cutset component and cycle component of the flows, with a trainable model for the inverse of the flow function in the CR. These components are summed and output as the flow prediction, for downstream use.

the edge flow space $\mathbb{R}^{|\mathcal{E}|}$: given the incidence matrix $B \in \{-1, 0, 1\}^{|\mathcal{V}|\times|\mathcal{E}|}$, the *cycle space* $\ker(B)$ and *cutset space* $\operatorname{Img}(B^{\mathsf{T}})$ are orthogonal, and $\mathbb{R}^{|\mathcal{E}|} = \ker(B) \oplus \operatorname{Img}(B^{\mathsf{T}})$. We refer the reader to [59, §9.4] for a primer on cycle and cutset spaces. Accordingly, we decompose the vector $f = \mathrm{FN}_{h,\theta}(G, u)$ as $f = f_{\text{cyc}} + f_{\text{cut}}$, where $f_{\text{cyc}} \in \ker(B)$ and $f_{\text{cut}} \in \operatorname{Img}(B^{\mathsf{T}})$. The cutset component is readily determined from (3.2.3), since $Bf = Bf_{\text{cut}} = u$ implies that $f_{\text{cut}} = B^{\dagger}u$. Then we must analyze (3.2.4) to solve for f_{cyc}. Define a *cycle projection matrix* $P \in \mathbb{R}^{m\times m}$ as the oblique projection onto $\ker(B)$ parallel to $\operatorname{Img}([a]B^{\mathsf{T}})$:

$$P = I_m - [a]B^{\mathsf{T}} \left(B[a]B^{\mathsf{T}}\right)^{\dagger} B \tag{3.2.5}$$

Based on this projection, we define a map $T : \ker(B) \to \ker(B)$ for all $f_{\text{cyc}} \in \ker(B)$ by

$$T(f_{\text{cyc}}) = P \left(f_{\text{cyc}} - d_{\min}[a]h_{\theta}^{-1}([a]^{-1}f_{\text{cyc}} + [a]^{-1}B^{\dagger}u)\right) \tag{3.2.6}$$

We can show that f_{cyc} is the unique fixed point of T, and that T is a contraction mapping, leading to a simple algorithm to compute this fixed point.

Theorem 3.2.2 (Properties of T)**.** *Consider an implicit flow network with parameters $d_{\min}$, $d_{\max}$, and θ, with flow functions h_θ. Suppose that the inputs $G = (\mathcal{V}, \mathcal{E}, a)$ and $u \in \mathbb{1}_{|\mathcal{V}|}^{\perp}$ are given, and let $B \in \{-1, 0, 1\}^{|\mathcal{V}|\times|\mathcal{E}|}$ be the signed incidence matrix of G. The following are true:*

(i) T is a contraction mapping with respect to $||\cdot||_{2,[a]^{-1}}$, with Lipschitz constant

$$\text{Lip}(T) \leq 1 - \frac{d_{\min}}{d_{\max}},$$

(ii) the sequence of iterates $f_{\text{cyc}}^{(k+1)} = T(f_{\text{cyc}}^{(k)})$ starting from any initial condition $f_{\text{cyc}}^{(0)} \in \ker(B)$ converges to a unique fixed point f_{cyc},

(iii) the output of the implicit flow network is unique and given by

$$\text{FN}_{h,\theta}(G, u) = f_{\text{cyc}} + B^{\dagger}u \tag{3.2.7}$$

Consequently, IFN is well-posed.

Proof: To prove statement (i), choose any $f_{\text{cyc}} \in \ker(B)$, let $y = [a]^{-1}f_{\text{cyc}} + [a]^{-1}B^{\dagger}u$ for brevity, and observe that

$$\begin{aligned}\left|\left|\frac{\partial T(f_{\text{cyc}})}{\partial f_{\text{cyc}}}\right|\right|_{2,[a]^{-1}} &= \left|\left|[a]^{-\frac{1}{2}}P\left(I_m - d_{\min}[a]\frac{\partial h_\theta^{-1}(y)}{\partial y}[a]^{-1}\right)[a]^{\frac{1}{2}}\right|\right|_2 \\ &\leq ||[a]^{-\frac{1}{2}}P[a]||_2\left|\left|[a]\left(I_m - d_{\min}[a]\frac{\partial h_\theta^{-1}(y)}{\partial y}[a]^{-1}\right)[a]^{\frac{1}{2}}\right|\right|_2 \\ &= \left|\left|[a]^{-\frac{1}{2}}\left(I_m - d_{\min}[a]\frac{\partial h_\theta^{-1}(y)}{\partial y}[a]^{-1}[a]^{-1}\right)[a]^{\frac{1}{2}}\right|\right|_2\end{aligned}$$

where $||[a]^{-\frac{1}{2}}P[a]^{\frac{1}{2}}||_2 = 1$ because $[a]^{-\frac{1}{2}}P[a]^{\frac{1}{2}}$ is a symmetric and idempotent matrix, i.e., an orthogonal projection. Then

$$\left|\left|\frac{\partial T(f_{\text{cyc}})}{\partial f_{\text{cyc}}}\right|\right|_{2,[a]^{-1}} = \max_{e\in\mathcal{E}} \left|1 - d_{\min}(h_\theta^{-1})'(y_e)\right| \leq 1 - \frac{d_{\min}}{d_{\max}}$$

Hence

$$\text{Lip}(T) = \sup_{f_{\text{cyc}}\in\mathbb{R}^m} \left|\left|\frac{\partial T(f_{\text{cyc}})}{\partial f_{\text{cyc}}}\right|\right|_{2,[a]^{-1}} \leq 1 - \frac{d_{\min}}{d_{\max}} < 1$$

Then statement (ii) follows from statement (i) and the Banach fixed point theorem. To prove statement (iii), observe that $f_{\text{cyc}} = Pf_{\text{cyc}}$, so $f_{\text{cyc}} = T(f_{\text{cyc}})$ if and only if

$$P[a]h_\theta^{-1}\left([a]^{-1}f\right) = \mathbb{0}_m \tag{3.2.8}$$

where $f = f_{\text{cyc}} + B^\dagger u$. But $\ker(P[a]) = \text{Img}(B^\top)$, so (3.2.8) is equivalent to the existence of $x \in \mathbb{R}^n$ such that

$$h^{-1}([a]^{-1}f) = B^\top x \tag{3.2.9}$$

and (3.2.9) is equivalent to (3.2.4). ■

Theorem 3.2.2 provides a simple algorithm for computing the IFN output f: pick any $f_{\text{cyc}}^{(0)} \in \ker(B)$, repeatedly apply the map T until approximate convergence, then add $B^\dagger u$. Some care is required when implementing this map. Since P is a dense matrix with $|\mathcal{E}|^2$ entries, it is undesirable to explicitly construct the cycle space projection matrix for large networks. Instead, in order to project a vector $v \in \mathbb{R}^{|\mathcal{E}|}$, we can use the fact that

$$w \triangleq (B[a]B^\top)^\dagger Bv = \arg\min_{w\in\mathbb{R}^n} \left\{||B[a]B^\top w - Bv||_2\right\}$$

so the projection is evaluated as $Pv = v - [a]B^\top w$. Using this method of projection to implement T, the fixed point iteration to compute $\text{FN}_{h,\theta}(G,u)$ is stated in Algorithm 3.

Algorithm 3 Evaluating the implicit flow network.

```
1:  $B \leftarrow$ signed incidence matrix of $G$
2:  $f_{\text{cut}} \leftarrow \arg\min_{f_{\text{cut}} \in \mathbb{R}^m} \{||Bf_{\text{cut}} - u||_2\}$
3:  $f_{\text{cyc}} \leftarrow \mathbb{0}_m$
4:  $\Delta f_{\text{cyc}} \leftarrow \infty \mathbb{1}_m$
5:  while $||\Delta f_{\text{cyc}}||_{2,[a]^{-1}} > \epsilon$ do
6:      $v \leftarrow d_{\min}[a] h_\theta^{-1}\left([a]^{-1} f_{\text{cyc}} + [a]^{-1} f_{\text{cut}}\right)$
7:      $w \leftarrow \arg\min_{w \in \mathbb{R}^n} \left\{||B[a]B^\mathsf{T} w - Bv||_2\right\}$
8:      $\Delta f_{\text{cyc}} \leftarrow v - [a]B^\mathsf{T} w$
9:      $f_{\text{cyc}} \leftarrow f_{\text{cyc}} - \Delta f_{\text{cyc}}$
10: end while
11: $f \leftarrow f_{\text{cyc}} + f_{\text{cut}}$
12: return $f$
```

Theorem 3.2.3 (Implicit Flow Networks, Forward Pass)**.** *Consider an implicit flow network with parameters $d_{\min}$, $d_{\max}$, and θ, with flow functions h_θ. Suppose that the inputs $G = (\mathcal{V}, \mathcal{E}, a)$ and $u \in \mathbb{1}_{|\mathcal{V}|}^\perp$ are given, and let $B \in \{-1, 0, 1\}^{|\mathcal{V}| \times |\mathcal{E}|}$ be the signed incidence matrix of G. The following are true of Algorithm 3, with a tolerance of $\epsilon > 0$:*

(i) for each iteration $k = 1, 2, \ldots$ of the loop, let $f_{\text{cyc}}^{(k)}$ represent the new value of f_{cyc} defined on line 9; and let $f_{\text{cyc}}^{(0)} = \mathbb{0}_m$. Then

$$f_{\text{cyc}}^{(k+1)} = T(f_{\text{cyc}}^{(k)}), \quad \forall k \geq 0;$$

(ii) the algorithm converges with at most k^* iterations of the while loop, where

$$k^* = 1 + \frac{\log\left(d_{\min}^{-1} \rho^{-1} \epsilon\right)}{\log\left(1 - \frac{d_{\min}}{d_{\max}}\right)} \tag{3.2.10}$$

and $\rho = ||[a]^{\frac{1}{2}} h_\theta^{-1}([a]^{-1} B^\dagger u)||_2$; and *(iii) the algorithm returns* $f \in \mathbb{R}^{|\mathcal{E}|}$*, where*

$$||f - \mathrm{FN}_{h,\theta}(G, u)||_{2,[a]^{-1}} \leq \left(\frac{d_{\max} - d_{\min}}{d_{\min}}\right) \epsilon \tag{3.2.11}$$

Proof: To prove statement (i), let $k \geq 0$ and consider iteration $k+1$ of the loop. The iteration first defines $v = d_{\min}[a]h_\theta^{-1}\left([a]^{-1}f_{\mathrm{cyc}}^{(k)} + [a]^{-1}f_{\mathrm{cut}}\right)$ on line 6. Then on line 7,

$$w = \underset{w \in \mathbb{R}^n}{\arg\min} \left\{||B[a]B^\mathsf{T}w - Bv||_2\right\} = \left(B[a]B^\mathsf{T}\right)^\dagger Bv$$

and line 8 defines

$$\Delta f_{\mathrm{cyc}} = v - [a]B^\mathsf{T}w = \left(I_m - [a]B^\mathsf{T}\left(B[a]B^\mathsf{T}\right)^\dagger\right)v = Pv$$

Finally, on line 9,

$$f_{\mathrm{cyc}}^{(k+1)} = f_{\mathrm{cyc}}^{(k)} - Pv = f_{\mathrm{cyc}}^{(k)} - d_{\min}P[a]h_\theta^{-1}\left([a]^{-1}f_{\mathrm{cyc}}^{(k)} + [a]^{-1}f_{\mathrm{cut}}\right)$$

A simple inductive argument shows that $f_{\mathrm{cyc}}^{(k)} \in \ker(B)$. The base case $f_{\mathrm{cyc}}^{(0)} = \mathbb{0}_m$ is trivial, for all $k' \geq 0$, line 9 ensures that $f_{\mathrm{cyc}}^{(k'+1)} \in \ker(B)$ so long as $f_{\mathrm{cyc}}^{(k')} \in \ker(B)$. Hence $f_{\mathrm{cyc}}^{(k)} = Pf_{\mathrm{cyc}}^{(k)}$, and we conclude that

$$f_{\mathrm{cyc}}^{(k+1)} = P\left(f_{\mathrm{cyc}}^{(k)} - d_{\min}[a]h_\theta^{-1}\left([a]^{-1}f_{\mathrm{cyc}}^{(k)} + [a]^{-1}f_{\mathrm{cut}}\right)\right) = T(f_{\mathrm{cyc}}^{(k)})$$

To prove statement (ii), recall from Theorem 3.2.2 that $\mathrm{Lip}(T) \leq 1 - d_{\max}^{-1}d_{\min}$, which (together with statement (i)) implies that, for all $k \geq 0$,

$$\begin{aligned}
||f_{\text{cyc}}^{(k+1)} - f_{\text{cyc}}^{(k)}||_{2,[a]^{-1}} &\leq \left(1 - \frac{d_{\min}}{d_{\max}}\right)^k ||f_{\text{cyc}}^{(1)} - f_{\text{cyc}}^{(0)}||_{2,[a]^{-1}} \\
&= d_{\min}\left(1 - \frac{d_{\min}}{d_{\max}}\right)^k ||P[a]h_\theta^{-1}\left([a]^{-1}B^\dagger u\right)||_{2,[a]^{-1}} \\
&= d_{\min}\left(1 - \frac{d_{\min}}{d_{\max}}\right)^k \rho
\end{aligned}$$

The algorithm terminates after iteration k if and only if $||f_{\text{cyc}}^{(k)} - f_{\text{cyc}}^{(k-1)}||_{2,[a]^{-\frac{1}{2}}} \leq \epsilon$, so the algorithm will have terminated after k^* iterations if

$$d_{\min}\left(1 - \frac{d_{\min}}{d_{\max}}\right)^{k^*-1} \rho \leq \epsilon$$

which is equivalent to

$$k^* \geq 1 + \frac{\log\left(d_{\min}^{-1}\rho^{-1}\epsilon\right)}{\log\left(1 - \frac{d_{\min}}{d_{\max}}\right)}$$

Finally, to prove statement (iii), note that the algorithm terminates after iteration k as soon as

$$||f_{\text{cyc}}^{(k)} - f_{\text{cyc}}^{(k-1)}||_{2,[a]^{-1}} \leq \epsilon$$

If f_{cyc} is the true fixed point of T, then using a general property of contraction mappings,

$$\begin{aligned}
||f_{\text{cyc}}^{(k)} - f_{\text{cyc}}||_{2,[a]^{-1}} &\leq \frac{\text{Lip}(T)}{1 - \text{Lip}(T)}||f_{\text{cyc}}^{(k)} - f_{\text{cyc}}^{(k-1)}||_{2,[a]^{-1}} \\
&\leq \left(\frac{d_{\max} - d_{\min}}{d_{\min}}\right)\epsilon
\end{aligned}$$

Therefore, the vector f returned by the algorithm satisfies

$$||f - \text{FN}_{h,\theta}(G,u)||_{2,[a]^{-1}} = ||f_{\text{cyc}}^{(k)} - f_{\text{cyc}}||_{2,[a]^{-1}} \leq \left(\frac{d_{\max} - d_{\min}}{d_{\min}}\right)\epsilon$$

■

If evaluating h_θ^{-1} is sufficiently simple, then the most expensive step in the iteration is solving the ordinary least squares problem on line 7. Using a general-purpose solver, the complexity of this operation is roughly $O(|\mathcal{V}|^3)$. But $B[a]B^\intercal$ is a sparse Laplacian matrix, so we can use a specialized Laplacian solver that reduces the complexity to $O(|\mathcal{E}| \log^k |\mathcal{E}|)$ for some constant k [87].

The bound on the number of iterations k^* can be computed before any forward pass, since evaluating h_θ^{-1} does not require solving the IFN equations. But we can further simplify the bound by approximating $h_\theta^{-1}(0) = 0$, which is often justified because physical flow functions generally have a root at the origin. Using the fact that $(h_\theta^{-1})'(y) \leq d_{\min}^{-1} y$, we can then eliminate the dependence on h_θ^{-1}:

$$k^* \leq 1 + \log\left(1 - \frac{d_{\min}}{d_{\max}}\right)\left(\log \epsilon - \log\left(||[a]^{-\frac{1}{2}} B^\dagger u||_2\right)\right)$$

3.2.2 Computing the Gradients

In order to train the flow function and any upstream models, it is necessary to backpropagate gradients through the IFN layer. We can perform this backward pass using implicit differentiation, and it turns out that the gradients of $\mathrm{FN}_{h,\theta}(G, u)$ with respect to the parameters θ, a, and u can also be computed using Algorithm 3, i.e., by writing the gradient as the output of an auxiliary implicit flow network.

Theorem 3.2.4 (Gradients)**.** *Consider an implicit flow network with parameters $d_{\min}$, $d_{\max}$, and θ, with flow functions h_θ. Suppose that the inputs $G = (\mathcal{V}, \mathcal{E}, a)$ and $u \in \mathbb{1}_{|\mathcal{V}|}^\perp$ are given, and let $B \in \{-1, 0, 1\}^{|\mathcal{V}| \times |\mathcal{E}|}$ be the signed incidence matrix of G. Let $f = \mathrm{FN}_{h,\theta}(G, u)$, and let w be a scalar entry of θ, a, or u. We can compute the derivatives $\frac{df}{dw}$ as follows.*

Define a vector of flow functions $g : \mathbb{R}^{|\mathcal{E}|} \to \mathbb{R}^{|\mathcal{E}|}$ *by*

$$g(\eta) = \mathcal{D}^{-1}\left(\eta - [a]^{-1}\frac{\partial v}{\partial w}\right), \quad \forall \eta \in \mathbb{R}^{|\mathcal{E}|} \tag{3.2.12}$$

where $\mathcal{D} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{E}|}$ *is the diagonal matrix with entries*

$$\mathcal{D}_{ee} = \left.\frac{dh_\theta^{-1}(y_e)}{dy_e}\right|_{y_e = a_e^{-1} f_e}, \quad \forall e \in \mathcal{E} \tag{3.2.13}$$

and $v = [a]h_\theta^{-1}([a]^{-1}f_{\mathrm{cyc}} + [a]^{-1}B^\dagger u)$*. Then*

$$\frac{df}{dw} = \mathrm{FN}_{g,\cdot}(G, \mathbb{0}_n) + B^\dagger \frac{du}{dw} \tag{3.2.14}$$

(We use the notation $\cdot$ *in place of* θ*, since* g *has no trainable parameters.) Furthermore, the derivative constraint parameters* $d_{\min}, d_{\max}$ *from the original implicit flow network are valid for the new implicit flow network.*

Proof: Let $v = [a]h_\theta^{-1}([a]^{-1}f_{\mathrm{cyc}} + [a]^{-1}B^\dagger u)$. From Theorem 3.2.2, we can write $f = f_{\mathrm{cyc}} + B^\dagger u$, where f_{cyc} is the unique fixed point of T. Therefore $\frac{df}{dw} = \frac{df_{\mathrm{cyc}}}{dw} + B^\dagger \frac{du}{dw}$, so the remainder of the proof is to show that $\frac{df_{\mathrm{cyc}}}{dw} = \mathrm{FN}_{g,\cdot}(G, \mathbb{0}_n)$.

Since $f_{\mathrm{cyc}} = T(f_{\mathrm{cyc}})$, and $Pf_{\mathrm{cyc}} = f_{\mathrm{cyc}}$, we have

$$f_{\mathrm{cyc}} = P\left(f_{\mathrm{cyc}} - d_{\min} v\right) = f_{\mathrm{cyc}} - d_{\min} P v$$

so an equivalent characterization of f_{cyc} is the unique solution to the equations

$$Bf_{\mathrm{cyc}} = \mathbb{0}_n$$

$$Pv = \mathbb{0}_m$$

Since

$$\frac{dv}{dw} = \frac{\partial v}{\partial w} + \frac{\partial v}{\partial f_{\text{cyc}}}\frac{df_{\text{cyc}}}{dw} = \frac{\partial v}{\partial w} + \mathcal{D}\frac{df_{\text{cyc}}}{dw}$$

then differentiating and factoring out $[a]$, we obtain

$$\begin{aligned} B\frac{df_{\text{cyc}}}{dw} &= \mathbb{0}_n \\ P[a]\left([a]^{-1}\frac{\partial v}{\partial w} + [a]^{-1}\mathcal{D}\frac{df_{\text{cyc}}}{dw}\right) &= \mathbb{0}_m \end{aligned}$$

Since $\ker(P[a]) = \operatorname{Img}(B^\mathsf{T})$, there exists $x \in \mathbb{R}^n$ such that

$$[a]^{-1}\frac{\partial v}{\partial w} + [a]^{-1}\mathcal{D}\frac{df_{\text{cyc}}}{dw} = B^\mathsf{T}x$$

which we can re-write as

$$\frac{df_{\text{cyc}}}{dw} = [a]\mathcal{D}^{-1}\left(B^\mathsf{T}x - [a]^{-1}\frac{\partial v}{\partial w}\right) = [a]g(B^\mathsf{T}x)$$

Hence $\frac{df_{\text{cyc}}}{dw}$ is the solution to

$$B\frac{df_{\text{cyc}}}{dw} = \mathbb{0}_n \tag{3.2.15}$$

$$\frac{df_{\text{cyc}}}{dw} = [a]g(B^\mathsf{T}x) \tag{3.2.16}$$

which is identical to (3.2.3)–(3.2.4) with $\frac{df_{\text{cyc}}}{dw}$ in place of f, $\mathbb{0}_n$ in place of u, and g in place of h_θ. Furthermore, g respects the same $d_{\min}, d_{\max}$ derivative constraints as h_θ, since for each $e \in \mathcal{E}$,

$$g_e'(\eta_e) = \frac{1}{\mathcal{D}_{ee}} = \left.\frac{dh_\theta(y_e)}{dy_e}\right|_{y_e = a_e^{-1}f_e} \in [d_{\min}, d_{\max}]$$

It follows that $\frac{df_{\text{cyc}}}{dw}$ is the output of the implicit flow network with flow functions g and parameters $d_{\min}, d_{\max}$, evaluated on the original graph G and nodal demands $\mathbb{0}_n$. ■

In other words, to compute the gradient with respect to a parameter, we perform a single evaluation of the implicit flow network. In order to compute the derivatives with respect to some parameter or input w, we first evaluate the partial derivatives $\frac{\partial v}{\partial w}$ and the total derivatives $\frac{du}{dw}$. Then we construct the flow functions g according to (3.2.12), and solve an implicit flow network to find $\frac{df}{dw}$ according to (3.2.14). It is easy to evaluate $\frac{du}{dw}$, but for convenience, we provide the values of $\frac{\partial v}{\partial w}$ below:

$$\frac{\partial v}{\partial \theta_i} = [a]\frac{dh_\theta^{-1}([a]^{-1}f)}{d\theta_i}, \qquad \frac{\partial v}{\partial a_e} = \operatorname{diag}\left(h_\theta^{-1}([a]^{-1}f) - [a]^{-1}\mathcal{D}f\right)_e, \qquad \frac{\partial v}{\partial u_i} = \left(\mathcal{D}B^\dagger\right)_i$$

3.3 Comparison with Optimization Models

The state-of-the-art methods for flow estimation [51] and [52] use an optimization problem to predict flows. The approach presented in Chapter 2 (based on [40]) falls within the same category. After a suitable transformation to incorporate external flow injections u, we can state this optimization problem as

$$\hat{f} = \underset{f\in\mathbb{R}^{|\mathcal{E}|}}{\arg\min}\left\{||f||_{2,[q]}^2 + \lambda^2||Bf-u||_2^2 \text{ s.t. } f_e = \tilde{f}_e,\ \ \forall \text{ labeled edges } e\in\mathcal{E}\right\} \tag{3.3.1}$$

where $\lambda > 0$, and $q > \mathbb{0}_m$ is some vector of edge weights. In [51], $q = \mathbb{1}_m$, while [52, 40] allows q to be the output of a neural network. IFN is not explicitly an optimization problem, but it can be cast as one that is similar to (3.3.1):

Theorem 3.3.1 (Optimization Form of IFN). *Consider an IFN with flow function h_θ. Suppose that the inputs $G = (\mathcal{V}, \mathcal{E}, a)$ and $u \in \mathbb{1}_{|\mathcal{V}|}^\perp$ are given, and let $B \in \{-1,0,1\}^{|\mathcal{V}|\times|\mathcal{E}|}$ be the signed incidence matrix of G. Then the IFN output can be stated as the solution* *of a convex optimization problem:*

$$\mathrm{FN}_{h,\theta}(G,u) = \operatorname*{arg\,min}_{f\in\mathbb{R}^m} \left\{ \sum_{e\in\mathcal{E}} \int_0^{f_e} h_\theta^{-1}(a_e^{-1}z)\ dz\ \ s.t.\ Bf = u \right\} \tag{3.3.2}$$

Proof: Since h_θ^{-1} is increasing, the optimization problem in (3.3.2) has a convex cost function with linear constraints, so the KKT conditions are necessary and sufficient. Letting $x \in \mathbb{R}^m$ be a vector of Lagrange multipliers, the Lagrangian is

$$\mathcal{L} = \sum_{e\in\mathcal{E}} \int_0^{f_e} h_\theta^{-1}(a_e^{-1}z)\ dz - x^\intercal (Bf - u)$$

leading to the stationarity condition

$$\mathbb{0}_m^\intercal = \frac{\partial \mathcal{L}}{\partial f} = h_\theta^{-1}(f^\intercal [a]^{-1}) - x^\intercal B$$

which is equivalent to (3.2.4). Additionally, the primal constraint $Bf = u$ is equivalent to (3.2.3), so the minimizer of the optimization problem is identical to the output of the IFN. ■

Theorem 3.3.1 can be interpreted as a nonlinear generalization of the Thomson principle from electrical circuits theory [55]. Interestingly, the theorem sets up a direct comparison between IFN and the models in [51, 52, 40]. If the flow function h_θ is the identity map, then (3.3.2) can be simplified as

$$\mathrm{FN}_{h,\theta}(G,u) = \operatorname*{arg\,min}_{f\in\mathbb{R}^m} \left\{ \sum_{e\in\mathcal{E}} ||f||_{2,[a]^{-1}}^2 \text{ s.t. } Bf = u \right\} \tag{3.3.3}$$

Ignoring the constraints from labeled flows, we can interpret (3.3.1) as using a penalty method to approximate the output of an IFN with a linear flow function. Thus, we have three distinct differences between IFN and the optimization-based approaches. First, IFN allows for a nonlinear flow function, while [51, 52, 40] implicitly assume a linear CR. Second, IFN imposes flow conservation as a hard constraint rather than an approximate constraint (which is a limitation if u is uncertain). Finally, IFN does not incorporate flow measurements directly; rather, the model exploits these measurements during training to learn the proper flow function (and train any upstream models for the IFN inputs), making it less sensitive to noise in the labeled flows.

3.4 Models for Flow Functions

In order to implement an IFN, it is necessary to parameterize its inverse flow function h_θ^{-1}. Since the flow function is essentially a trainable activation function, i.e., a scalar nonlinearity, simple models are likely to be sufficient. The main difficulty with selecting a flow function is that its slope must be bounded by $d_{\min} \leq h'_\theta(y) \leq d_{\max}$ for all $y \in \mathbb{R}$. This section proposes a simple scalar nonlinearity that is guaranteed to respect arbitrary upper and lower bounds on its slope.

Definition 3.4.1 (Derivative-Constrained Perceptron)**.** Let $k \in \mathbb{Z}_{>0}$ be a hidden layer size, let $a, b, c \in \mathbb{R}^k$ be freely trainable parameters (encoded within the parameter vector θ), and let σ be a non-expansive activation. Let $p, q \geq 1$ such that $p^{-1} + q^{-1} = 1$, and let $\bar{d}_{\min} \leq \bar{d}_{\max} \in \mathbb{R}$. Then the *derivative-constrained perceptron* $\mathrm{N}(x, \theta)$ is the 3-layer neural network defined by

$$\bar{c}(\theta) = \left(1 - \frac{(||c||_p||a||_q - 1)_+}{||c||_p||a||_q}\right) c \tag{L1}$$

$$\mathrm{N}_0(x, \theta) = \bar{c}^\mathsf{T}(\theta)\sigma(ax + b) \tag{L2}$$

$$\mathrm{N}(x, \theta) = \left(\frac{\bar{d}_{\max} - \bar{d}_{\min}}{2}\right) \mathrm{N}_0(x, \theta) + \left(\frac{\bar{d}_{\max} + \bar{d}_{\min}}{2}\right) x \tag{L3}$$

Intuitively, (L1) re-scales c so that the perceptron in (L2) is guaranteed to be non-expansive in x. Then (L3) re-centers and re-scales the derivatives of the perceptron from the range $[-1, 1]$ to $[\bar{d}_{\min}, \bar{d}_{\max}]$.

Theorem 3.4.2 (Derivative-Constrained Perceptron)**.** *Let* $\mathrm{N}(x, \theta)$ *be a derivative-constrained perceptron with* $\bar{d}_{\min} \leq \bar{d}_{\max} \in \mathbb{R}$*. Then for all parameter values* θ*,*

$$\bar{d}_{\min} \leq \frac{d}{dx} \mathrm{N}(x, \theta) \leq \bar{d}_{\max}, \quad \forall x \in \mathbb{R} \tag{3.4.1}$$

Proof: Due to (L3), it is clear that the derivative bounds (3.4.1) hold if and only if

$$\left| \frac{\mathrm{dN}_0(x, \theta)}{\mathrm{d}x} \right| \leq 1, \qquad \forall x \in \mathbb{R} \tag{3.4.2}$$

For all $x, x' \in \mathbb{R}$, by Hölder's inequality,

$$\begin{aligned} |\mathrm{N}_0(x, \theta) - \mathrm{N}_0(x', \theta)| &= \left| \bar{c}^{\mathsf{T}}(\theta) \left(\sigma(ax + b) - \sigma(ax' + b) \right) \right| \\ &\leq ||\bar{c}(\theta)||_p \, ||\sigma(ax + b) - \sigma(ax' + b)||_q \end{aligned}$$

Since σ is non-expansive, its Lipschitz constant with respect to the q-norm is

$$\mathrm{Lip}(\sigma) = \sup_{\eta \in \mathbb{R}^k} \left|\left| \frac{\partial \sigma(\eta)}{\partial \eta} \right|\right|_q = \sup_{\eta_0 \in \mathbb{R}} |\sigma'(\eta_0)| \leq 1$$

and thus

$$||\sigma(ax + b) - \sigma(ax' + b)||_q \leq ||a(x - x')||_q \leq ||a||_q |x - x'|$$

Furthermore, by (L1),

$$\begin{aligned}||\bar{c}(\theta)||_p||a||_q &= \left(1 - \frac{(||c||_p||a||_q - 1)_+}{||c||_p||a||_q}\right)||c||_p||a||_q \\ &= ||c||_p||a||_q - (||c||_p||a||_q - 1)_+ \\ &= \min\{1, ||c||_p||a||_q\}\end{aligned}$$

so that

$$|\mathrm{N}_0(x,\theta) - \mathrm{N}_0(x',\theta)| \leq \min\{1, ||c||_p||a||_q\}\,|x - x'| \leq |x - x'|$$

for all x, x'. Hence (3.4.2) is satisfied. ■

Note that the values $\bar{d}_{\text{min}}, \bar{d}_{\text{max}}$ in Definition 3.4.1 and Theorem 3.4.2 are distinct from the IFN parameters $d_{\text{min}}, d_{\text{max}}$. Since we parameterize the *inverse* flow function h_θ^{-1} in IFN, one shoudl set $\bar{d}_{\text{min}} = d_{\text{max}}^{-1}$ and $\bar{d}_{\text{max}} = d_{\text{min}}^{-1}$ to implement h_θ^{-1} with a derivative-constrained perceptron.

3.5 Numerical Experiments

We studied the transductive task of predicting unlabeled flows, given that some labeled flows in the same network are known. If the edges $\mathcal{E}$ are partitioned into a labeled set $\mathcal{E}_l$ and an unlabeled set $\mathcal{E}_u$, the task is to predict the missing flows $\{f_e : e \in \mathcal{E}_u\}$ given the labeled flows $\{f_e : e \in \mathcal{E}_l\}$. For each network, we randomly selected a fraction of the edges to be labeled edges, and we trained IFN and baselines on the labeled edges. Then we evaluated the RMSE of the flows predicted for the unlabeled edges $\mathcal{E}_u$ to compute the testing error. See Appendix A in the supplementary material for full details. Code is available at `https://github.com/KevinDalySmith/implicit-flow-networks`.

3.5.1 Datasets

AC Power We selected 6 standard power network test cases. The first 4 test cases (IEEE-57, IEEE-118, IEEE-145, and IEEE-300) are synthetic transmission system test cases, while the remaining cases ACTIVSg200 and ACTIVSg500 are similar to the Illinois and South Carolina power grids, respectively [88]. Each test case contains the topology and electrical parameters of the power network, as well as baseline demands and power injections at each node. While branch resistances are typically small, we set them to zero to ensure lossless transmission lines. We used the MATPOWER toolbox [89] to solve the power flow equations, then recorded the active power flows on each branch (f), computed the net active power injections at each node (u), and selected relevant electrical parameters as edge attributes (series reactance, tap ratio, and voltage magnitude at the two incident nodes).

Water Distribution We selected 3 sample water distribution networks from the ASCE Task Committee on Research Databases for Water Distribution Systems database [90], representing municipal water distribution systems in Fairfield, CA, Bellingham, WA, and Harrisburg, PA. Each network contains the topology of the distribution system, as well as the characteristics of pipes and other network elements and nodal demands. We used the WNTR package [91] to compute the flow rates through each pipe (f), net inflow rate at each node (u), and edge weights associated with each pipe.

3.5.2 Models and Experiment Details

IFN Architecture In order to use the IFN layer to predict power flows, we created a two-layer model. The first layer estimates positive edge weights $a \in \mathbb{R}^{|\mathcal{E}|}$ according to $a_e = \exp(\mathrm{L}(z_e))$ for all $e \in \mathcal{E}$, where L is a linear module, and z_e is the log-transformed vector of edge attributes. The second layer is an IFN. To predict water flows, we used an IFN layer alone, supplying the edge weights from the dataset as input (rather than learning them from other edge attributes). For both water and power, the IFN layer uses a derivative-constrained perceptron as the inverse flow function ($k = 128$, $p = q = \frac{1}{2}$) with a ReLU activation function. For power, we set $d_{\text{min}} = 0.4$ and $d_{\text{max}} = 2$; and for water, $d_{\text{min}} = 0.2$ and $d_{\text{max}} = 20$.

Baselines We compared the IFN model against four baselines. The minimum divergence method (*Div*) from [51] minimizes the nodal divergence $||Bf||_2^2$ and a regularization term $\lambda||f||_2^2$. The bilevel optimization methods from [52] replace the uniform regularizer with a weighted regularizer $||f||_{2,[q]}^2$, where q is a vector of weights. In *Bil-MLP* and *Bil-GCN*, q is the output of either a 2-layer MLP or GCN model with edge attributes as inputs (we use 64 nodes in each hidden layer with ReLU activations). In *Bil-True*, we specify q as the reciprocal of the coefficient in the linearized CR for AC power networks, so that *Bil-True* approximates (3.3.3) with a as the ground-truth edge weight. For water experiments, *Bil-True* uses the same edge weights as the IFN model.

All of the baselines assume that nodal divergence Bf should be approximately zero, but nodes in power networks inject and withdraw power according to the supply / demand vector u, resulting in nonzero divergence. Thus, when we evaluate the baselines, we transform the power network into a divergence-free network by introducing a "source node", adding an edge from the source node to all nodes in $\mathcal{V}$, and treating the entries of u as the flows along each corresponding virtual edge.

3.5.3 Results

Figure 3.2 reports the results for the AC power networks, and Figure 3.3 reports the results for water distribution systems. In both types of networks, the IFN model

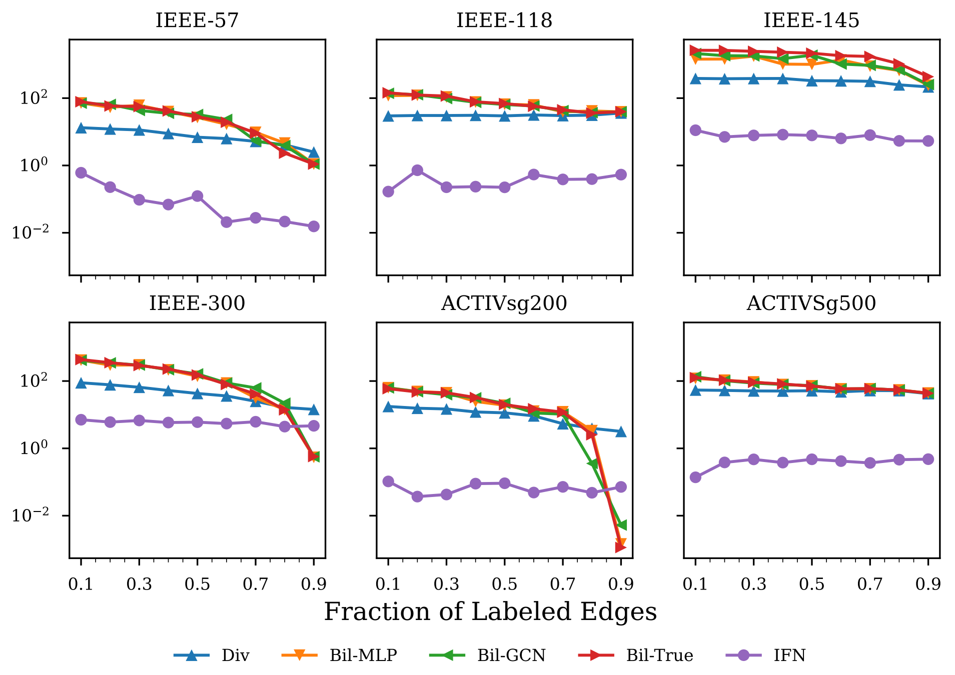

Figure 3.2: Results for missing flow prediction in AC power networks. Reported values are the RMSE (in units of MW) on the testing set, averaged across 10 trials. Note the vertical axis is in a log scale.

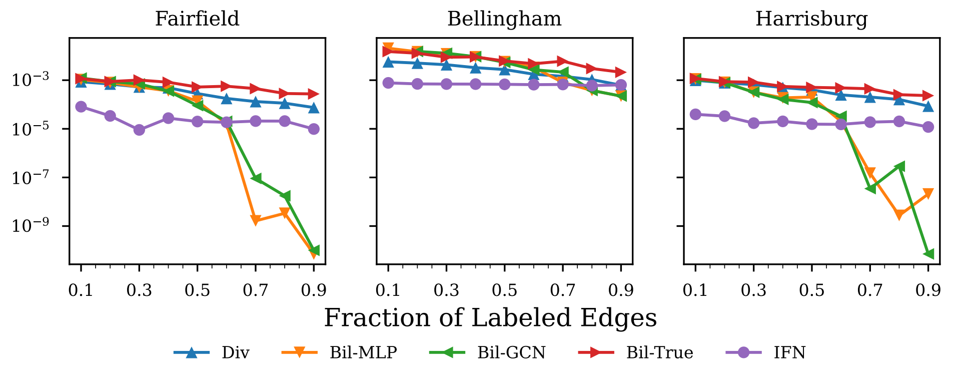

Figure 3.3: Results for missing flow prediction in water distribution systems. Reported values are the RMSE (in units of m^3/s) on the testing set, averaged across 10 trials.

significantly outperforms the baselines on all of the networks when a small fraction of edges are labeled (less than 80% in power and less than 60% in water). While the other baselines tend to improve as more labeled edges are made available for training, IFN achieves near-optimal performance with as few as 10% of the edges labeled.

Chapter 4

Conclusion and Future Directions

4.1 Thesis Summary

In this dissertation, we proposed two end-to-end architectures and their corresponding learning algorithms to infer missing flows in infrastructure networks based on partial observations. Both algorithms belong to a broad family of machine learning models called implicit neural networks.

As opposed to existing works, the first framework is directly based on classical results in the circuits theory literature and can handle operational constraints, greatly benefiting its explainability. Such a framework is highly modular, and integrates Graph Neural Networks and a physics-inspired Implicit Layer. Through extensive comparisons with state-of-the-art approaches, we have empirically shown that our framework achieves higher accuracy. Moreover, we discussed scenarios where employing the Implicit Layer leads to a substantial speedup in training compared to the state of the art. While our experiments focused on a transductive setting, we anticipate strong performance in inductive scenarios as well, which are pertinent to the infrastructure networks domain.

As for the second model, we focused on the scenario where a) all the network supplies and demands are known, and b) the infrastructure is characterized by a nonlinear constitutive relationship (CR). The proposed implicit model incorporates physics through a conservation law and through the existence of a (latent) constitutive relationship between flows and nodal variables. We have demonstrated that a simple architecture using this model can learn to accurately predict active power flows in AC networks and water distribution systems.

The two proposed architectures target different aspects of the flow estimation problem, and share some similarities in terms of how the physics is embedded, and how they give rise to non-traditional implicit neural network layers.

4.2 Future Directions

A limitation of our approaches consists of considering the equilibrium scenario only: Networks with dynamic states (that are potentially available as a stream of data) are subject of ongoing research. Finally, the proposed approaches are limited to lossless networks; adapting them to networks with losses constitutes a compelling research direction.

Future work may investigate more elaborate architectures using the implicit layers as a layer, wherein the supply / demand vector, edge weights, or even the graph itself could be predicted from upstream models, and the flows themselves used for downstream tasks. Another interesting extension may be to extend our methods to networks with higher-order interactions, i.e., hypergraphs [92] and simplicial complexes [93, 94].

Specifically to IFN, some additional limitations should also be addressed in future work. IFN assumes that the graph is undirected, which does not adequately model networks with unidirectional flows (e.g., traffic) or lossy flows (e.g., resistive power grids). IFN also assumes a CR that depends on the *difference* between nodal variables. This form appears frequently in physical systems, but in other network flow models (like Daganzo traffic models [54]), the CR has a more general dependence on the nodal variables. These limitations may be addressed with extensions of IFN's contraction algorithm.

Appendix A

Experiments Details on IFN

A.1 AC Power Datasets

We created datasets from 6 AC power network test cases. Each dataset that we created represents a snapshot of an AC power network in its steady state, consisting of four components: the network topology (as an oriented, undirected graph), four attributes on each edge (voltage magnitude at the two incident nodes, series reactance, and tap ratio), the net power injection at each node, and the active power flow through each branch.

Original Data We generated our datasets using MATPOWER, an open-source toolbox for power system simulation in MATLAB [89]. The toolbox includes many standard test

Test Case	MATPOWER Case Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$
IEEE-57	`case57`	57	135
IEEE-118	`case118`	118	297
IEEE-145	`case145`	145	567
IEEE-300	`case300`	300	709
ACTIVSg200	`case_ACTIVSg200`	200	445
ACTIVSg500	`case_ACTIVSg500`	500	1084

Table A.1: MATPOWER test case details.

cases, which contain a network topology and tables of electrical and economic parameters for each bus (node), branch (edge), and generator. We selected 6 test cases, listed in Table A.1. The raw data files for these test cases are available from the MATPOWER source[1], and details on the test case file format are contained in Appendix B of the user manual[2].

Data Generation After loading each test case into MATPOWER, we performed the following two modifications of the network parameters:

(i) We set branch resistances (column 3 in the branch data table) to zero, so that transmission lines in the system are lossless. This step was necessary because IFN is limited to undirected graphs, while lossy lines are more appropriately modeled with a pair of directed edges, since the power injected at one endpoint does not equal the power withdrawn from the other endpoint. Fortunately, branch resistances are typically small before this modification.

(ii) We replaced any negative series reactances (column 4 in the branch data table) with a positive value, chosen as the median of the positive series reactances in the same network. We performed this modification because negative series reactances results in *decreasing* constitutive relationships on the corresponding edges, whereas IFN assumes that the constitutive relationship is increasing. This modification only affected two networks: IEEE-145, in which 24 (4.2%) of the branches were assigned a series reactance of 0.2306; and IEEE-300, in which 1 (0.1%) of the branches was assigned a series reactance of 0.059.

We then computed the resulting power flows using the `runpf` function and recorded the results.

[1] https://github.com/MATPOWER/matpower/tree/master/data
[2] https://matpower.org/docs/MATPOWER-manual.pdf

Pre-Processing Finally, we converted the results from the MATPOWER simulation into a PyTorch Geometric data object, with the following attributes:

- `edge_index`, the edge index tensor, containing the topology from the test case.
- `x`, a tensor of net active power injections at each node, which has the property that $\mathbb{1}_n^\mathsf{T} x = 0$. (This tensor is identical to the supply / demand vector u in the paper.)
- `edge_attr`, a tensor of four relevant attributes for each edge: the voltage magnitudes at the two incident nodes, the series reactance, and the tap ratio.
- `f_true`, the tensor of active power flows on each edge simulated by MATPOWER.

The net active power injections at each node are computed according to

$$u_i = \mathrm{PG}_i - \mathrm{PD}_i - \mathrm{GS}_i \mathrm{VM}_i^2$$

where PG_i is active power generated at i, PD_i is active power demanded, GS_i is shunt conductance, and VM_i is the voltage magnitude.

A.2 Water Distribution Dataset

We created 3 datasets representing snapshots of municipal water distribution networks in their steady state, consisting of four components: the network topology (as an oriented, undirected graph), weights for each edge, the net inflow rates at each node, and the flow rate through each pipe.

Original Data Each of the datasets is based on a network from the ASCE Task Committee on Research Databases for Water Distribution Systems database [90]. Networks in this database contain a distribution network topology and tables of hydraulic parameters

Test Case	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$
Fairfield	111	125
Bellingham	121	162
Harrisburg	261	286

Table A.2: Water distribution network details.

and operating characteristics for each node, pipe (edge), pump, reservoir, and storage tank in the network. We selected 3 networks, listed in Table A.2 and plotted in Figure A.1. The raw data files are available online[3].

Data Generation and Preprocessing We loaded each network INP file into WNTR and ran the WNTR simulator with a hydraulic accuracy of 10^{-8}. We then converted the results into a PyTorch Geometric data object, with the following attributes:

- `edge_index`, the edge index tensor, containing the topology from the test case.
- `x`, a tensor of net inflows at each node, which has the property that $\mathbb{1}_n^\mathsf{T} x = 0$.
- `edge_attr`, a tensor of three relevant attributes for each edge: the pipe length, pipe diameter, and pipe roughness coefficient.
- `f_true`, the tensor of flow rates through each pipe simulated by WNTR.

Edge weights are computed according to the formula

$$a_e = (0.27855) C_e D_e^{2.63} L_e^{-0.54} \tag{A.2.1}$$

where C_e is the roughness coefficient (unitless), D_e is the diameter (meters), and L_e is the pipe length (meters) [4].

[3] http://www.uky.edu/WDST/index.html
[4] https://wntr.readthedocs.io/en/latest/hydraulics.html

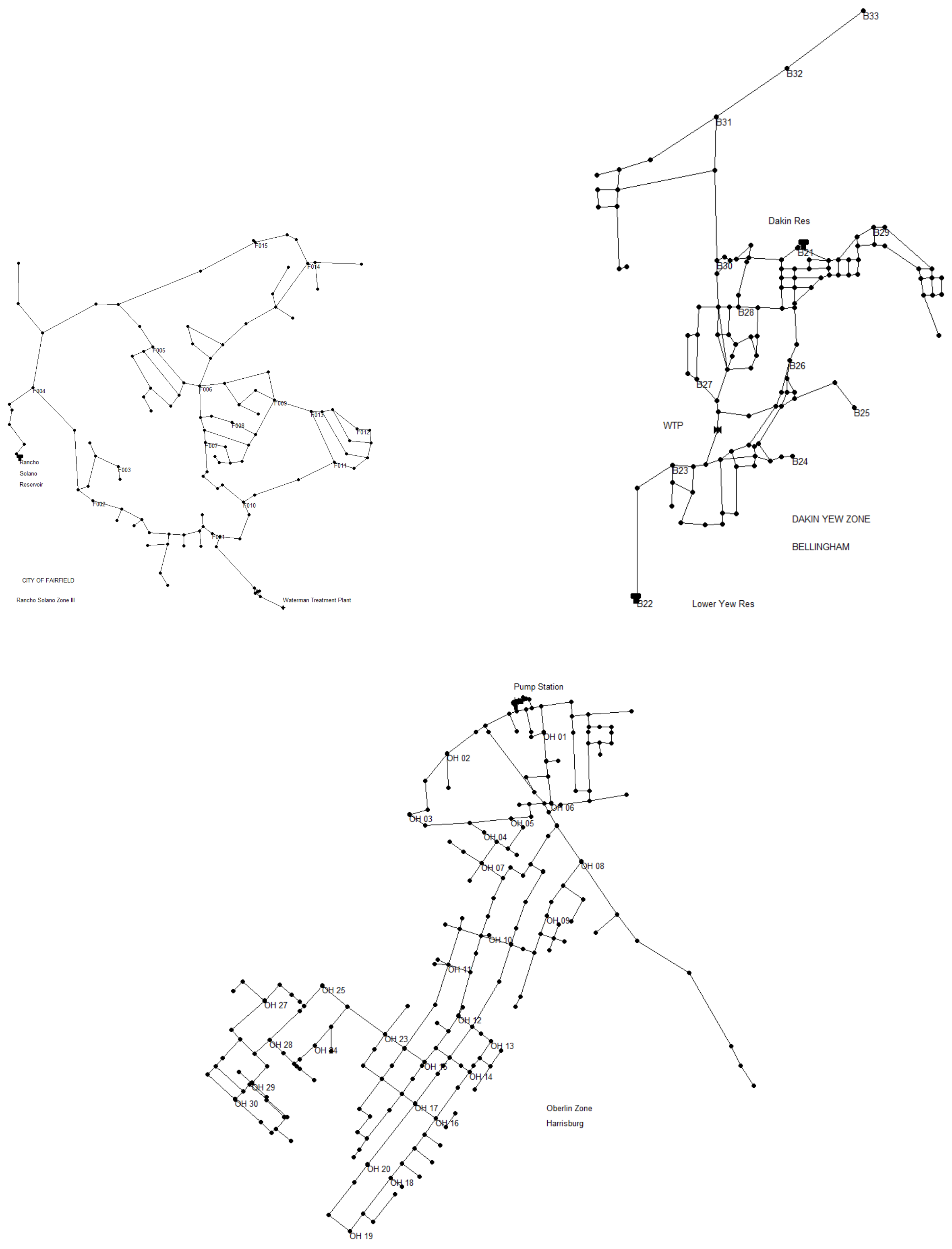

Figure A.1: Network maps of the three water distribution systems: Fairfield (upper left), Bellingham (upper right), and Harrisburg (bottom).

A.3 Details on IFN

Our IFN implementation uses Algorithm 3 to compute the layer's forward pass. We set the maximum number of iterations in this algorithm to 100, with a tolerance of $\epsilon = 10^{-2}$ for power and $\epsilon = 10^{-4}$ for water. With the release of PyTorch 1.11.0, the `torch.linalg.lstsq` method[5] now supports automatic differentiation, allowing PyTorch to automatically backpropagate through the Algorithm 3 iterations, instead of using Theorem 3.2.4. We found that Algorithm 3 terminated with a small enough number of iterations that automatic differentiation was faster, so we opted to use this rather than the method from Theorem 3.2.4. We trained the IFN models to minimize the RMSE loss function by minimizing the RMSE loss function

$$\ell_{\text{rmse}} = \sqrt{\frac{1}{|\mathcal{E}_l|} \sum_{e \in \mathcal{E}} (f_e - \text{FN}_{h,\theta}(G, u)_e)^2}$$

A.4 Details on Baselines

Analogously to the baselines implementation in [40], we implemented all of the baselines by adapting Silva's code[6] from [52], refactoring some utility functions to decrease runtime. Following [52], we perform the following two data normalization steps:

(i) negative flows are converted into positive flows by flipping the orientation of the corresponding edges and replacing the entries of `f_true` with their absolute value, and

(ii) flows are proportionally normalized to the range $[0, 1]$ within each network.

After training with the normalized flows and computing the missing flow predictions, the

[5] https://pytorch.org/docs/stable/generated/torch.linalg.lstsq.html

[6] https://openreview.net/forum?id=l0V53bErniB

predictions are denormalized before computing the testing RMSE.

Div The minimizing divergence baseline from [51] has a single hyperparameter, λ, from the regularization term $\lambda^2||f||_2^2$ in the loss function. We set $\lambda = 0.1$ for all networks and fractions of labeled edges by hand-tuning the parameter to the proper order of magnitude.

Bilevel Baselines All three of the bilevel baselines (Bil-MLP, Bil-GCN, and Bil-True) have several hyperparameters related to the bilevel optimization algorithm. For most of these parameters, we use the same settings as [52]: the number of iterations for the inner optimization problem is 300 during training and 3000 during evaluation, and the number of k-fold cross validation folds is 10; however, we increased the number of iterations of the outer optimization problem from 10 to 100, with an early stopping interval of 10, to ensure that the outer optimization problem was given sufficient time to converge. As with [52], we used a 2-layer MLP and GCN in Bil-MLP and Bil-GCN, respectively, but we increased the size of the hidden layer to 64.

Bil-True Like IFN, the baselines Bil-MLP and Bil-GCN train a model to predict edge weights from side information (if we interpret $\mathcal{Q}$ as a diagonal matrix of edge weights). We devised Bil-True as a third baseline to use the "ground-truth edge weights" instead of training a model. For water experiments, these ground-truth edge weights are given by (A.2.1). For the power experiments, we compute these edges weights from the AC active power flow equation: in a lossless AC power grid, active power flows f_{ij} on each edge $\{i, j\} \in \mathcal{E}$ are given by

$$f_{ij} = \frac{v_i v_j}{x_{ij}\tau_{ij}} \sin(\theta_i - \theta_j) \approx \frac{v_i v_j}{x_{ij}\tau_{ij}} (\theta_i - \theta_j) \tag{A.4.1}$$

where v_i, v_j are the voltage magnitudes on incident nodes, x_{ij} is the series reactance, τ_{ij} is the tap ratio, and θ_i, θ_j are the incident voltage angles. Since (A.4.1) is the constitutive relationship for AC power networks, examining its linear approximation in light of (3.3.3) suggests using $x_{ij}\tau_{ij}/v_i v_j$ as the regularizer weight on f_{ij}.

A.5 Details on Training

We trained all models in a Google Colab notebook, using the Adam optimizer. We used an initial learning rate of 0.01, which we found to have good performance for all of the models. Training was terminated when the training loss had not decreased for 10 epochs.

Bibliography

[1] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, 2004.

[2] N. Zucchet and J. Sacramento, *Beyond backpropagation: bilevel optimization through implicit differentiation and equilibrium propagation*, *Neural Computation* **34** (2022), no. 12 2309–2346.

[3] B. Colson, P. Marcotte, and G. Savard, *An overview of bilevel optimization*, *Annals of Operations Research* **153** (2007), no. 1 235–256.

[4] H. von Stackelberg, *Market Structure and Equilibrium.* Springer Berlin Heidelberg, 2010.

[5] L. Vicente, G. Savard, and J. Júdice, *Descent approaches for quadratic bilevel programming*, *Journal of optimization theory and applications* **81** (1994), no. 2 379–399.

[6] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, *Forward and reverse gradient-based hyperparameter optimization*, in *International Conference on Machine Learning*, pp. 1165–1173, PMLR, 2017.

[7] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, *Bilevel programming for hyperparameter optimization and meta-learning*, in *International Conference on Machine Learning*, vol. 80, pp. 1568–1577, PMLR, 2018.

[8] A. Lopes da Silva, *Notes on bilevel optimization*, *unpublished notes* (2020).

[9] P. Vicol, J. P. Lorraine, F. Pedregosa, D. Duvenaud, and R. B. Grosse, *On implicit bias in overparameterized bilevel optimization*, in *International Conference on Machine Learning*, pp. 22234–22259, PMLR, 2022.

[10] B. Amos and J. Z. Kolter, *OptNet: Differentiable optimization as a layer in neural networks*, in *International Conference on Machine Learning*, vol. 70, pp. 136–145, PMLR, 2017.

[11] S. Barratt, *On the differentiability of the solution to convex optimization problems*, 2019.

[12] W. Rudin, *Principles of Mathematical Analysis.* International Series in Pure and Applied Mathematics. McGraw-Hill, 3 ed., 1976.

[13] M. Claesen and B. De Moor, *Hyperparameter search in machine learning*, *arXiv preprint arXiv:1502.02127* (2015).

[14] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*, vol. 112. Springer, 2013.

[15] S. Keerthi, V. Sindhwani, and O. Chapelle, *An efficient method for gradient-based adaptation of hyperparameters in svm models*, *Advances in neural information processing systems* **19** (2006).

[16] G. Kunapuli, K. P. Bennett, J. Hu, and J.-S. Pang, *Classification model selection via bilevel programming*, *Optimization Methods & Software* **23** (2008), no. 4 475–489.

[17] F. Pedregosa, *Hyperparameter optimization with approximate gradient*, in *International conference on machine learning*, pp. 737–746, PMLR, 2016.

[18] C. Finn, P. Abbeel, and S. Levine, *Model-agnostic meta-learning for fast adaptation of deep networks*, in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017.

[19] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, *et. al.*, *Model-based reinforcement learning: A survey*, *Foundations and Trends® in Machine Learning* **16** (2023), no. 1 1–118.

[20] R. Bellman, *Dynamic Programming.* Princeton Univ Press, Princeton, NJ, 1957. Reprint: [95].

[21] J. R. Taylor, *Classical Mechanics.* University Science Books, 2005.

[22] D. V. Schroeder, *An Introduction to Thermal Physics.* Oxford University Press, 01, 2021.

[23] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, *A tutorial on energy-based learning*, *Predicting structured data* **1** (2006), no. 0.

[24] A. Mas-Colell, M. D. Whinston, and J. R. Green, *Microeconomic Theory.* Oxford University Press, New York, 1995.

[25] A. Davydov, V. Centorrino, A. Gokhale, G. Russo, and F. Bullo, *Contracting dynamics for time-varying convex optimization*, 2023.

[26] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems.* Cambridge University Press, 2017.

[27] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory and Design.* Nob Hill Publishing, 2 ed., 2019.

[28] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, *Differentiable MPC for end-to-end planning and control*, in *Advances in Neural Information Processing Systems*, 2018.

[29] X. Xiao, T. Zhang, K. M. Choromanski, T.-W. E. Lee, A. Francis, J. Varley, S. Tu, S. Singh, P. Xu, F. Xia, S. M. Persson, D. Kalashnikov, L. Takayama, R. Frostig, J. Tan, C. Parada, and V. Sindhwani, *Learning model predictive controllers with real-time attention for real-world navigation*, in *Proceedings of The 6th Conference on Robot Learning* (K. Liu, D. Kulic, and J. Ichnowski, eds.), vol. 205 of *Proceedings of Machine Learning Research*, pp. 1708–1721, PMLR, 14–18 Dec, 2022.

[30] J. Drgona, K. Kis, A. Tuor, D. Vrabie, and M. Klauco, *Differentiable predictive control: Deep learning alternative to explicit model predictive control for unknown nonlinear systems*, *Journal of Process Control* **116** (2022) 80–92.

[31] A. Hauswirth, S. Bolognani, G. Hug, and F. Dorfler, *Timescale separation in autonomous optimization*, *IEEE Transactions on Automatic Control* **66** (2021), no. 2 611–624.

[32] G. Bianchin, J. Cortés, J. I. Poveda, and E. Dall'Anese, *Time-varying optimization of LTI systems via projected primal-dual gradient flows*, *IEEE Transactions on Control of Network Systems* **9** (2022), no. 1 474–486.

[33] F. Bullo, *Contraction Theory for Dynamical Systems.* Kindle Direct Publishing, 1.1 ed., 2023.

[34] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, *Differentiable convex optimization layers*, in *Advances in Neural Information Processing Systems*, 2019.

[35] M. Blondel, Q. Berthet, M. Cuturi, R. Frostig, S. Hoyer, F. Llinares-López, F. Pedregosa, and J.-P. Vert, *Efficient and modular implicit differentiation*, *arXiv preprint arXiv:2105.15183* (2021).

[36] E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, and S. Chintala, *Generalized inner loop meta-learning*, *arXiv preprint arXiv:1910.01727* (2019).

[37] L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Q. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson, J. Dong, B. Amos, and M. Mukadam, *Theseus: A library for differentiable nonlinear optimization*, 2023.

[38] J. Ren, X. Feng, B. Liu, X. Pan, Y. Fu, L. Mai, and Y. Yang, *Torchopt: An efficient library for differentiable optimization*, *arXiv preprint arXiv:2211.06934* (2022).

[39] A. Tuor, J. Drgona, J. Koch, M. Shapiro, D. Vrabie, and S. Briney, *NeuroMANCER: Neural Modules with Adaptive Nonlinear Constraints and Efficient Regularizations*, .

[40] F. Seccamonte, A. K. Singh, and F. Bullo, *Inference of infrastructure network flows via physics-inspired implicit neural networks*, in *2023 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 1040–1045, 2023.

[41] A. K. Singh, Ibraheem, S. Khatoon, M. Muazzam, and D. K. Chaturvedi, *Load forecasting techniques and methodologies: A review*, in *2012 2nd International Conference on Power, Control and Embedded Systems*, IEEE, 2012.

[42] J.-S. Brouillon, E. Fabbiani, P. Nahata, F. Dörfler, and G. Ferrari-Trecate, *Bayesian methods for the identification of distribution networks*, in *IEEE Conference on Decision and Control*, pp. 3646–3651, 2021.

[43] S. G. Vrachimis, D. G. Eliades, and M. M. Polycarpou, *Leak detection in water distribution systems using hydraulic interval state estimation*, in *IEEE Conference on Control Technology and Applications (CCTA)*, pp. 565–570, 2018.

[44] B. S. Rego, S. G. Vrachimis, M. M. Polycarpou, G. V. Raffo, and D. M. Raimondo, *State estimation and leakage detection in water distribution networks using constrained zonotopes*, *IEEE Transactions on Control Systems Technology* **30** (2022), no. 5 1920–1933.

[45] D. B. Work, S. Blandin, O.-P. Tossavainen, B. Piccoli, and A. M. Bayen, *A Traffic Model for Velocity Data Assimilation*, *Applied Mathematics Research eXpress* **2010** (04, 2010) 1–35.

[46] B. Yu, H. Yin, and Z. Zhu, *Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting*, in *International Joint Conference on Artificial Intelligence*, p. 3634–3640, AAAI Press, 2018.

[47] Y. Li, R. Yu, C. Shahabi, and Y. Liu, *Diffusion convolutional recurrent neural network: Data-driven traffic forecasting*, in *International Conference on Learning Representations*, 2018.

[48] A. A. Ahmadi and B. E. Khadir, *Learning dynamical systems with side information*, in *Learning for Dynamics and Control*, vol. 120, pp. 718–727, PMLR, 2020.

[49] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, *Physics-informed machine learning*, *Nature Reviews Physics* **3** (2021), no. 6 422–440.

[50] E. Inanc, Y. Gurses, A. Habboush, Y. Yildiz, and A. M. Annaswamy, *Neural network adaptive control with long short-term memory*, 2023.

[51] J. Jia, M. T. Schaub, S. Segarra, and A. R. Benson, *Graph-based semi-supervised & active learning for edge flows*, in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, p. 761–771, July, 2019.

[52] A. Silva, F. Kocayusufoglu, S. Jafarpour, F. Bullo, A. Swami, and A. K. Singh, *Combining physics and machine learning for network flow estimation*, in *International Conference on Learning Representations*, (Online), May, 2021.

[53] C. A. Desoer and E. S. Kuh, *Basic Circuit Theory.* IEEE Press, 1969.

[54] G. Como, *On resilient control of dynamical flow networks*, *Annual Reviews in Control* **43** (2017) 80–90.

[55] P. G. Doyle and J. L. Snell, *Random Walks and Electric Networks.* Mathematical Association of America, 1984.

[56] D. Duvenaud, Z. J. Kolter, and M. Johnson, *Deep implicit layers tutorial - neural ODEs, deep equilibirum models, and beyond.*, in *Advances in Neural Information Processing Systems, Tutorial*, 2020.

[57] F. Harary and R. Z. Norman, *Some properties of line digraphs*, *Rendiconti del Circolo Matematico di Palermo* **9** (1960) 161–168.

[58] L. Gong and Q. Cheng, *Exploiting edge features for graph neural networks*, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9203–9211, 2019.

[59] F. Bullo, *Lectures on Network Systems.* Kindle Direct Publishing, 1.6 ed., Jan., 2022.

[60] T. Brown, J. Hörsch, and D. Schlachtberger, *PyPSA: Python for Power System Analysis*, *Journal of Open Research Software* **6** (2018), no. 4.

[61] S. W. Fung, H. Heaton, Q. Li, D. McKenzie, S. Osher, and W. Yin, *Jfb: Jacobian-free backpropagation for implicit models*, *Proceedings of the AAAI Conference on Artificial Intelligence* (2022).

[62] K. D. Smith, F. Seccamonte, A. Swami, and F. Bullo, *Physics-informed implicit representations of equilibrium network flows*, in *Advances in Neural Information Processing Systems*, Nov., 2022.

[63] L. R. Ford and D. R. Fulkerson, *Maximal flow through a network*, in *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[64] M. Lippi, M. Bertini, and P. Frasconi, *Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning*, *IEEE Transactions on Intelligent Transportation Systems* **14** (2013), no. 2 871–882.

[65] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. Li, *Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction*, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 5668–5675, 2019.

[66] L. Böttcher, H. Wolf, B. Jung, P. Lutat, M. Trageser, O. Pohl, X. Tao, A. Ulbig, and M. Grohe, *Solving ac power flow with graph neural networks under realistic constraints*, in *2023 IEEE Belgrade PowerTech*, pp. 1–7, 2023.

[67] S. Bai, J. Z. Kolter, and V. Koltun, *Deep equilibrium models*, in *Advances in Neural Information Processing Systems*, 2019.

[68] E. Winston and J. Z. Kolter, *Monotone operator equilibrium networks*, in *Advances in Neural Information Processing Systems*, 2020.

[69] M. Revay, R. Wang, and I. R. Manchester, *Lipschitz bounded equilibrium networks*, .

[70] L. El Ghaoui, F. Gu, B. Travacca, A. Askari, and A. Tsai, *Implicit deep learning*, *SIAM Journal on Mathematics of Data Science* **3** (2021), no. 3 930–958.

[71] S. Jafarpour, A. Davydov, A. V. Proskurnikov, and F. Bullo, *Robust implicit networks via non-Euclidean contractions*, in *Advances in Neural Information Processing Systems*, Dec., 2021.

[72] S. W. Fung, H. Heaton, Q. Li, D. McKenzie, S. Osher, and W. Yin, *Fixed point networks: Implicit depth models with Jacobian-free backprop*, 2021. ArXiv e-print.

[73] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, *Neural ordinary differential equations*, in *Advances in Neural Information Processing Systems*, 2018.

[74] H. Heaton, D. McKenzie, Q. Li, S. W. Fung, S. J. Osher, and W. Yin, *Learn to predict equilibria via fixed point networks*, *CoRR* **abs/2106.00906** (2021) [arXiv:2106.0090].

[75] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, *Geometric deep learning: Grids, groups, graphs, geodesics, and gauges*, *arXiv preprint arXiv:2104.13478* (2021).

[76] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, *arXiv preprint arXiv:1609.02907* (2016).

[77] W. Hamilton, Z. Ying, and J. Leskovec, *Inductive representation learning on large graphs*, *Advances in neural information processing systems* **30** (2017).

[78] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, *arXiv preprint arXiv:1710.10903* (2017).

[79] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, *Neural message passing for quantum chemistry*, in *International Conference on Machine Learning*, pp. 1263–1272, PMLR, 2017.

[80] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, and K. Kavukcuoglu, *Interaction networks for learning about objects, relations and physics*, in *Advances in Neural Information Processing Systems.* 2016.

[81] F. Gu, H. Chang, W. Zhu, S. Sojoudi, and L. El Ghaoui, *Implicit graph neural networks*, in *Advances in Neural Information Processing Systems*, 2020.

[82] P. Kundur, *Power System Stability and Control.* McGraw-Hill, 1994.

[83] E. J. Finnemore and J. B. Franzini, *Fluid mechanics with engineering applications.* McGraw-Hill, tenth ed., 2002.

[84] G. M. U. Din and A. K. Marnerides, *Short term power load forecasting using deep neural networks*, in *2017 International conference on computing, networking and communications (ICNC)*, pp. 594–598, IEEE, 2017.

[85] K. D. Smith and K. Studarus, *Limited-knowledge economic dispatch prediction using bayesian averaging of single-node models*, in *2018 IEEE International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*, IEEE, 2018.

[86] S. Jafarpour, E. Y. Huang, K. D. Smith, and F. Bullo, *Flow and elastic networks on the n-torus: Geometry, analysis and computation*, *SIAM Review* **64** (2022), no. 1 59–104.

[87] N. K. Vishnoi, *$Lx = b$, Laplacian solvers and their algorithmic applications*, *Theoretical Computer Science* **8** (2013), no. 1-2 1–141.

[88] A. B. Birchfield, T. Xu, K. M. Gegner, K. S. Shetye, and T. J. Overbye, *Grid structural characteristics as validation criteria for synthetic networks*, *IEEE Transactions on power systems* **32** (2016), no. 4 3258–3265.

[89] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, *MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education*, *IEEE Trans. Power Syst.* **26** (2011), no. 1 12–19.

[90] E. Hernadez, S. Hoagland, and L. Ormsbee, *Water distribution database for research applications*, in *World Environmental and Water Resources Congress 2016*, pp. 465–474, 2016.

[91] K. A. Klise, R. Murray, and T. Haxton, *An overview of the water network tool for resilience (WNTR)*, in *WDSA/CCWI Joint Conference Proceedings*, (Kingston,Ontario,Canada), July, 2018.

[92] J. Jo, J. Baek, S. Lee, D. Kim, M. Kang, and S. J. Hwang, *Edge representation learning with hypergraphs*, in *Advances in Neural Information Processing Systems* (M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), vol. 34, pp. 7534–7546, Curran Associates, Inc., 2021.

[93] T. M. Roddenberry, N. Glaze, and S. Segarra, *Principled simplicial neural networks for trajectory prediction*, in *International Conference on Machine Learning*, pp. 9020–9029, PMLR, 2021.

[94] M. Yang, E. Isufi, M. T. Schaub, and G. Leus, *Simplicial convolutional filters*, *IEEE Transactions on Signal Processing* **70** (2022) 4633–4648.

[95] R. Bellman, *Dynamic Programming.* Dover Publications, 2003. Reprint of 1972 sixth edition.