Wasserstein Distributionally Robust Graph Learning via Algorithm Unrolling

Xiang Zhang, *Student Member, IEEE*, Yinfei Xu, *Member, IEEE*, Mingjie Shao, *Member, IEEE*, and Yonina C. Eldar, *Fellow, IEEE*

***Abstract*—In this paper, we consider inferring the underlying graph topology from smooth graph signals. Most existing approaches learn graphs by minimizing a well-designed empirical risk using the observed data, which may be prone to data uncertainty that arises from noisy measurements and limited observability. Therefore, the learned graphs may be unreliable and exhibit poor out-of-sample performance. To enhance the robustness to data uncertainty, we propose a smoothness-based graph learning framework from a distributionally robust perspective, which is equivalent to solving an $\inf - \sup$ problem. However, learning graphs directly in this way is challenging since (i) the $\inf - \sup$ problem is intractable, and (ii) many parameters need to be manually determined. To address these issues, we first reformulate the $\inf - \sup$ problem into a tractable one, where robustness is achieved via a regularizer. Theoretically, we show that the regularizer can improve generalization of the proposed graph estimator by bounding the out-of-sample risks. We then propose an algorithm based on the ADMM framework to solve the induced problem and further unroll it into a neural network. All parameters are determined automatically and simultaneously by training the unrolled network. Extensive experiments on both synthetic and real-world data demonstrate that our approach can achieve superior and more robust performance than existing models on different observed signals.**

***Index Terms*—Graph learning, graph signal processing, distributionally robust optimization, Wasserstein distance, algorithm unrolling.**

Received 25 July 2023; revised 22 April 2024 and 7 November 2024; accepted 20 December 2024. Date of publication 23 January 2025; date of current version 10 February 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62371119, in part by Zhi Shan Young Scholar Program of the Southeast University, in part by the National Natural Science Foundation of China under Grant 62401340, and in part by the Natural Science Foundation of Shandong Province under Grant ZR2023QF103. The associate editor coordinating the review of this article and approving it for publication was Sheetal Kalyani. *(Corresponding author: Yinfei Xu.)*

Xiang Zhang and Yinfei Xu are with the School of Information Science and Engineering, Southeast University, Nanjing 210096, China (e-mail: xiangzhang369@seu.edu.cn; yinfeixu@seu.edu.cn).

Mingjie Shao is with the Key Laboratory of Systems and Control, Institute of Systems Science, Academy of Mathematics and Systems Science (AMSS), Chinese Academy of Sciences (CAS), Beijing 100149, China (e-mail: mingjieshao@amss.ac.cn).

Yonina C. Eldar is with Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 7610001, Israel (e-mail: yonina.eldar@weizmann.ac.il).

Digital Object Identifier 10.1109/TSP.2025.3526287

I. Introduction

RECENT years have witnessed a growing interest in signal processing and machine learning tasks involving graphs since the intrinsic information of structured data residing on topologically complicated domains can be flexibly represented with graphs [1]. Some celebrated graph-based models, such as spectral clustering [2] and graph neural networks [3], have achieved great success in many fields. However, among these models, the underlying graphs are built on prior knowledge that may not be available or accurate, affecting the performance of downstream graph-based tasks. Therefore, it is essential to learn graphs directly from the observed data on hand.

Inferring graph topology from raw data, also termed graph learning, has long been a research area of interest [4], [5]. Recently, with the rise of graph signal processing (GSP) [6], a class of widely studied models, referred to as smoothness-based models, attempt to learn graphs from the perspective of signal processing. These models postulate that the observed signals are smooth over the underlying graphs [7], [8]. Intuitively, a graph signal being smooth over a graph means that the signal values of two connected nodes should be similar. In practice, many signals appear to be smooth over the latent graphs, including temperature recordings [8] and medical data [9]. Smoothness-based models [7], [8], [10], [11] typically infer graph topology by solving a constrained Laplacian quadratic form minimization problem [12]. For example, [8] employs a squared Frobenius norm regularizer to control the sparsity of the learned graphs, and [7] utilizes a logarithmic barrier function to improve the overall graph connectivity. These constrained Laplacian optimization problems are equivalent to minimizing the empirical risks of observed data. As a result, the learned graphs inevitably depend heavily on the empirical distribution and are vulnerable to data uncertainty. Specifically, the empirical distribution may deviate from the true data distribution—which is unknown and depends on the underlying graphs—due to low-quality observations, leading to unreliable graphs that trouble downstream tasks. On the other hand, the graphs learned from empirical risk minimization (ERM) are prone to overfitting since they only exploit information in the observed data and ignore those unseen signals.

Here, we leverage ideas from distributionally robust optimization (DRO) [13] to learn graphs from smooth signals by considering data uncertainty. Specifically, we assume that the ground-truth data distribution is unknown but lies in an uncertainty set containing all distributions in the proximity of the empirical distribution. We then learn a graph by minimizing the worst-case expected risk of all distributions in the uncertainty set instead of the empirical risk. The graph learned in this way performs well under the unknown ground-truth distribution since it suppresses the risks of all distributions in the uncertainty set, which may also include the true distribution (if the uncertainty set is constructed properly).

Constructing the uncertainty set requires some metrics to measure distributional discrepancies. Common metrics in DRO include KL-divergence [14], moment-based metrics [15], and Wasserstein distance [16]. In this study, we employ the Wasserstein distance for the following reasons. (i) The Wasserstein uncertainty set contains richer distributions than those induced by other metrics, e.g., the moment metric [12], meaning it can handle more complex real-world distributions. (ii) Wasserstein distance is more statistically robust than other information divergences [17] and provides numerous theoretical results, which facilitates subsequent model analysis [16]. Two recent studies [18], [19] have applied Wasserstein distributionally robust optimization (WDRO) to infer graph topology. Our model differs from [18], [19] in that we learn graphs with Laplacian constraints, whereas [18], [19] learn precision matrices. Moreover, we theoretically analyze the out-of-sample (OOS) performance of the proposed model, which is not provided in [18], [19]. In parallel to our work, [12] also proposes to learn graphs from smooth signals under distributional uncertainty. However, it only uses the first two moments of data distributions to construct uncertainty sets, while our model uses Wasserstein distance, which may carry more information. Furthermore, we propose a method for constructing the uncertainty sets, which has not been explored before (see Section IV-B).

Nevertheless, it is challenging to learn graphs by minimizing the worst-case expected risk—which is equivalent to solving an $\inf - \sup$ problem—for two reasons. (i) The $\inf - \sup$ problem is intractable since it involves the expectation over all possible distributions in the uncertainty set. (ii) The uncertainty set size, together with other undetermined parameters, constitutes a large parameter search space, making it laborious to choose optimal parameters for the proposed framework.

Regarding the first challenge, many studies attempt to reformulate the $\inf - \sup$ problems induced by DRO into tractable forms [20], [21], [22]. However, the reformulation in the context of learning graphs from smooth signals has not been thoroughly studied. Inspired by [22], we reformulate our distributionally robust graph learning problem into a tractable one, where robustness is achieved via a regularization term whose weight is interpreted as the uncertainty set size. The reformulation provides an interpretable connection between robustness and regularizers. To further validate the necessity of the WDRO-induced regularizer, we analyze the OOS performance of the proposed model. The results illustrate that the regularizer contributes to bounding the OOS risks, which improves generalization of the proposed graph estimator.

The uncertainty set size needs to be carefully determined as it represents the level of robustness [19]. A large uncertainty set indicates higher confidence that the set contains the true distribution, while too large uncertainty sets may lead to over-conservative results [16]. A practical method to determine the uncertainty set size is cross-validation [21], which is computationally expensive. Moreover, there are other free parameters in our model, making it more difficult to use cross-validation due to the large search space. Some studies suggest a principle for choosing the uncertainty set size through concentration inequalities [16], [21]. Although computationally efficient, the sizes determined by this approach may be over-conservative [16]. The recent work [23] proposes a method based on empirical likelihood, which is unsuitable for our problem due to the additional Laplacian constraints. Thus, how to design an efficient method to automatically determine all parameters simultaneously, especially the uncertainty set size, remains unsolved.

To address the issue, we employ the algorithm unrolling (AU) technique [24] to transform the parameter selection task into a neural network training task. Algorithm unrolling offers a principled framework to express a conventional iterative algorithm as a neural network. It was first proposed in [25] to connect iterative algorithms for sparse coding to neural network architectures. Following [25], many studies attempt to unroll iterative algorithms, including compressive sensing [26], deconvolution [27], and stochastic control [28].

Recently, some works apply AU to GSP to denoise and restore graph signals [29], [30]. These approaches unroll graph signal denoising/restoration algorithms based on known graphs, while our goal is to use AU to learn graphs from signals. Some studies have also applied AU to graph learning problems. The GLAD model [31] unrolls an alternating minimization algorithm to learn graphs represented by precision matrices without Laplacian constraints. The recent work [9] learns graphs from smooth signals by unrolling a primal-dual splitting (PDS) algorithm. Our model, however, differs from that of the logarithmic barrier function in [9] and further accounts for data uncertainty. The latest work [32] leverages AU to learn graphs from convolutional mixtures, which is different from our smoothness-based model.

Here, we first propose an iterative algorithm based on the ADMM framework to solve the reformulated problem. Then, each iteration of the proposed algorithm is unrolled as a layer of a neural network, where the parameters to be determined are regarded as the trainable parameters. Moreover, we replace the handcrafted regularizers that endow topological properties with a multi-layer neural network. In this way, topological features are learned automatically from the data instead of being determined manually. The parameters, including the uncertainty set size, are learned through end-to-end training, and the trained networks are used to learn graphs. The merits of utilizing AU are two-fold. (i) The optimal parameters of the framework are simultaneously determined by training the network. As a result, the trained networks usually exhibit more powerful learning performance than traditional iterative algorithms with manually determined parameters. (ii) Compared to traditional neural networks, the unrolled network incorporates domain knowledge about smoothness-based graph learning models. Consequently, the unrolled network has more efficient parameters and requires fewer training data to extract information [24].

In summary, the contributions of this study are as follows:

- We propose a smoothness-based graph learning model from a distributionally robust perspective. Leveraging modern reformulation techniques, we reformulate the problem into a tractable form, where robustness is achieved via a regularizer whose weight is equivalent to the uncertainty set size. Our framework builds an interpretable bridge between robustness and graph regularizers. Furthermore, we theoretically analyze the OOS performance of the proposed graph estimator. The results reveal that the robustness-induced regularizer contributes to bounding the OOS risks, thereby mitigating overfitting. Thus, we corroborate the necessity of the regularizer in robust graph learning settings.
- We propose an ADMM algorithm to solve the induced problem and unroll it into a neural network to avoid excessive parameter searches. The parameters of the proposed framework, especially the uncertainty set size, are determined automatically and simultaneously by training the network.
- We conduct extensive experiments on both synthetic and real-world data to validate the effectiveness of the proposed framework. The experimental results demonstrate that our approach can achieve superior and more robust performance than state-of-the-art graph learning models on different observed signals.

Organization: The rest of this paper is organized as follows. Some background information is introduced in Section II. We propose our robust graph learning framework in Section III. Then, we develop an ADMM algorithm to solve the induced problem and further unroll it into a neural network in Section IV. Next, we conduct experiments on both synthetic data and real-world data in Section V. Finally, concluding remarks are presented in Section VI.

Notations: Throughout this paper, vectors, matrices, and sets are written in bold lowercase, bold uppercase letters, and calligraphic uppercase letters, respectively. Given a vector $\mathbf{y}$ and matrix $\mathbf{Y}$, $\mathbf{y}_{[i]}$ and $\mathbf{Y}_{[ij]}$ denote the i-th entry of $\mathbf{y}$ and the (i,j) entry of $\mathbf{Y}$, respectively. The vectors $\mathbf{1}$, $\mathbf{0}$, and matrix $\mathbf{I}$ represent all-one vectors, all-zero vectors, and identity matrices, respectively, whose dimensions depend on the context. The ℓ_q norm of a vector is denoted as $\|\cdot\|_q$, and $\|\cdot\|_{\mathrm{F}}$ represents Frobenius norm of a matrix. The notations $\dagger$, $\circ$, $\mathrm{Tr}(\cdot)$, $\mathrm{vec}(\cdot)$, and $\mathrm{diag}(\cdot)$ denote pseudo inverse, Hadamard product, trace operator, vectorization operator, and diagonalization, respectively. Given a distribution $\mathbb{P}$, we write $\mathbf{y}\sim\mathbb{P}$ to mean that the random vector $\mathbf{y}$ is distributed according to $\mathbb{P}$. Moreover, we use $\mathbb{E}_{\mathbf{y}\sim\mathbb{P}}[\cdot]$ to denote the expectation w.r.t. $\mathbb{P}$. Besides, $\Pr(\cdot)$ denotes taking probability. Finally, we use $\mathbb{S}^{d\times d}$ for the set of $d\times d$ symmetric matrices and $\mathbb{R}_+$ for set of nonnegative real values.

II. Background

A. Wasserstein Distance

The Wasserstein distance is a measure of the discrepancy between two distributions. Here, we define order-α Wasserstein distance as follows.

Definition 1 ([33]): For any $\alpha\in[1,\infty)$, order-α Wasserstein distance between two probability distributions $\mathbb{P}_1$ and $\mathbb{P}_2$ on $\mathbb{R}^d$ is defined as

$$W_\alpha(\mathbb{P}_1,\mathbb{P}_2) = \left(\inf_{\pi\in\mathcal{U}(\mathbb{P}_1,\mathbb{P}_2)}\int_{\mathbb{R}^d\times\mathbb{R}^d} C(\mathbf{a}_1,\mathbf{a}_2)^\alpha \pi(\mathrm{d}\mathbf{a}_1,\mathrm{d}\mathbf{a}_2)\right)^{\frac{1}{\alpha}}, \quad (1)$$

where $C(\cdot)$ is a cost function. In addition, $\mathcal{U}(\mathbb{P}_1,\mathbb{P}_2)$ represents the set of all joint distributions π on $\mathbf{a}_1,\mathbf{a}_2\in\mathbb{R}^d$, whose marginal distributions are $\mathbb{P}_1$ and $\mathbb{P}_2$, respectively.

We select the cost function $C(\cdot)$ as ℓ_p norm, i.e., $C(\mathbf{a}_1,\mathbf{a}_2)=\|\mathbf{a}_1-\mathbf{a}_2\|_p$, where $p\geq 1$. The definition indicates that the Wasserstein distance between two distributions represents the cost of an optimal mass transportation plan [33].

B. GSP Background

We consider undirected graphs with non-negative weights and no self-loops. For such a graph $\mathcal{G}=\{\mathcal{V},\mathcal{E}\}$ with d vertices, where $\mathcal{V}$ and $\mathcal{E}$ are the sets of nodes and edges respectively, its adjacency matrix $\mathbf{W}\in\mathbb{S}^{d\times d}$ is a symmetric matrix with zero diagonal entries and non-negative off-diagonal entries. The Laplacian matrix of $\mathcal{G}$ is defined as $\mathbf{L}=\mathbf{D}-\mathbf{W}$ [34], where the degree matrix $\mathbf{D}\in\mathbb{S}^{d\times d}$ is a diagonal matrix satisfying $\mathbf{D}_{[ii]}=\sum_{j=1}^d\mathbf{W}_{[ij]}$. The matrices $\mathbf{L}$ and $\mathbf{W}$ encode the topology of $\mathcal{G}$ since they have a one-to-one relationship. We study the graph signal $\mathbf{x}=\left[\mathbf{x}_{[1]},\ldots,\mathbf{x}_{[d]}\right]^\top\in\mathbb{R}^d$ associated with the graph $\mathcal{G}$, where $\mathbf{x}_{[i]}$ is the signal value of node i of $\mathcal{G}$. The smoothness of a graph signal $\mathbf{x}$ over $\mathcal{G}$ is defined as follows.

Definition 2 (Smoothness [8]): Given a graph signal $\mathbf{x}\in\mathbb{R}^d$ associated with the graph $\mathcal{G}$, the smoothness of $\mathbf{x}$ over $\mathcal{G}$ is defined as

$$\mathbf{x}^\top\mathbf{L}\mathbf{x}=\frac{1}{2}\sum_{i,j=1}^d \mathbf{W}_{[ij]}\left(\mathbf{x}_{[i]}-\mathbf{x}_{[j]}\right)^2, \quad (2)$$

where $\mathbf{L}$ and $\mathbf{W}$ are the Laplacian matrix and adjacency matrix of $\mathcal{G}$, respectively.

This quadratic form of (2) quantifies how much the signal $\mathbf{x}$ changes w.r.t. $\mathcal{G}$. A graph signal is said to be smooth over the corresponding graph if the value of (2) is small [8].

C. Smooth Signals Graph Learning

Given N observations $\mathbf{X}=[\mathbf{x}_1,\ldots\mathbf{x}_N]\in\mathbb{R}^{d\times N}$, the goal of smoothness-based graph learning is to infer the graph topology under the assumption that the signals $\mathbf{X}$ are smooth over the underlying graph $\mathcal{G}$ [8]. Formally, the problem is written as

$$\begin{aligned}\inf_{\mathbf{L}}\ &\frac{1}{N}\sum_{n=1}^N \mathbf{x}_n^\top\mathbf{L}\mathbf{x}_n + r(\mathbf{L})\\ \text{s.t. } &\mathbf{L}_{[ij]}=\mathbf{L}_{[ji]}<0,\ \text{for } i\neq j,\\ &\mathbf{L}\mathbf{1}=\mathbf{0},\\ &\mathrm{Tr}(\mathbf{L})=d, \end{aligned} \quad (3)$$

where $r(\mathbf{L})$ is a regularization term of $\mathbf{L}$ that endows $\mathcal{G}$ with desired properties such as sparsity. Some common choices of

$r(\mathbf{L})$ include $\kappa\|\mathbf{L}\|_{\mathrm{F}}^2$ [8] and $-\kappa\mathbf{1}^\top\log(\mathrm{diag}(\mathbf{L}))$ [7], where κ is the weight of the regularizers. The first two constraints of (3) are used to ensure the learned $\mathbf{L}$ is a valid Laplacian matrix. The last constraint is to avoid trivial solutions [8]. For notational simplicity, we define $\mathcal{L}$ as

$$\mathcal{L} \triangleq \left\{\mathbf{L} : \mathbf{L} \in \mathbb{S}^{d\times d}, \mathbf{L}\mathbf{1} = \mathbf{0},\, \mathbf{L}_{[ij]} \leq 0 \text{ for } i \neq j,\, \mathrm{Tr}(\mathbf{L}) = d\right\}.$$

Thus, problem (3) can be rephrased as

$$\inf_{\mathbf{L}\in\mathcal{L}} \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n^\top \mathbf{L}\mathbf{x}_n + r(\mathbf{L}). \tag{4}$$

However, the basic smoothness-based model (4) learns graphs directly from the observed signals $\mathbf{X}$, meaning that the performance may heavily depend on the observed data. To this end, in the next section, we propose a robust graph learning model grounded on (4) by considering data uncertainty of $\mathbf{X}$.

III. Wasserstein Distributionally Robust Graph Learning

In this section, we first propose the robust graph learning framework under the WDRO framework, which is next reformulated into a tractable problem. Finally, we analyze the OOS performance of the proposed model.

A. Basic Formulation

We first revisit (4) via the lens of ERM, which can be rewritten as

$$\inf_{\mathbf{L}\in\mathcal{L}} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_n}\left[\mathbf{x}^\top\mathbf{L}\mathbf{x}\right] + r(\mathbf{L}), \tag{5}$$

where $\mathbb{P}_n$ is the empirical distribution associated with the observed signals $\mathbf{x}_1, \ldots, \mathbf{x}_N$, i.e.,

$$\mathbb{P}_n = \frac{1}{N}\sum_{n=1}^{N}\mathrm{Dirac}(\mathbf{x}_n), \tag{6}$$

where $\mathrm{Dirac}(\mathbf{x}_n)$ denotes the Dirac point measure at data point $\mathbf{x}_n$. This formulation depends heavily on the empirical distribution, suggesting that it may suffer from the following disadvantages. (i) The empirical distribution may be far from the ground-truth due to limited observations and noisy measurements, making the learned graphs unreliable. (ii) The learned graphs are prone to overfitting as the model only exploits the information from the observed signals while ignoring unseen data [16]. Hence, we hope to learn a robust graph less affected by the sample quality.

To this end, we first construct a set

$$\mathcal{P} = \left\{\mathbb{P} :\ W_\alpha(\mathbb{P}, \mathbb{P}_n) \leq \epsilon\right\}, \tag{7}$$

where $W_\alpha(\mathbb{P}, \mathbb{P}_n)$, $\alpha \geq 1$, is the order-α Wasserstein distance between $\mathbb{P}$ and $\mathbb{P}_n$. As stated before, Wasserstein distance can bring tractable reformulation and an interpretable bridge between robustness and graph regularizers, which will be presented later. The uncertainty set $\mathcal{P}$ contains all distributions whose distances from the empirical distribution $\mathbb{P}_n$ are smaller than ϵ. We hereafter refer to ϵ as the uncertainty set size. We do not explicitly specify α here, as we will prove that it does not impact our formulation in the next subsection. We then learn a graph by minimizing the worst-case expected risk over all distributions in $\mathcal{P}$, which can be formulated as the following $\inf - \sup$ problem:

$$\inf_{\mathbf{L}\in\mathcal{L}} R(\mathbf{L};\mathbf{x}) + r(\mathbf{L}) = \inf_{\mathbf{L}\in\mathcal{L}}\sup_{\mathbb{P}\in\mathcal{P}} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}\left[\mathbf{x}^\top\mathbf{L}\mathbf{x}\right] + r(\mathbf{L}), \tag{8}$$

where

$$R(\mathbf{L};\mathbf{x}) = \sup_{\mathbb{P}\in\mathcal{P}} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}\left[\mathbf{x}^\top\mathbf{L}\mathbf{x}\right] \tag{9}$$

is called the worst-case expected risk. The philosophy underlying (8) is that if we minimize the worst-case expected risk, we can naturally push down the risks of all distributions in $\mathcal{P}$, which may include the true distribution $\mathbb{P}^*$ [21]. Thus, the expected risk of the learned graph, $\widehat{\mathbf{L}}$, over $\mathbb{P}^*$ is smaller than $R(\widehat{\mathbf{L}};\mathbf{x})$ (if $\mathcal{P}$ contains $\mathbb{P}^*$), indicating that $\widehat{\mathbf{L}}$ still performs well over the true distribution even if the empirical distribution deviates from the ground-truth. Intuitively, the uncertainty set size determines the probability that $\mathcal{P}$ contains $\mathbb{P}^*$. An uncertainty set with large ϵ is more likely to contain $\mathbb{P}^*$ as well as more nuisance distributions, making $\widehat{\mathbf{L}}$ less sensitive to unseen signals. However, if the ϵ is too large, $\mathcal{P}$ will contain too many "bad" distributions that may not be encountered in real life. As a result, the worst-case expected risk is unnecessarily high, resulting in the learned graphs being over-conservative. Therefore, ϵ is critical to the learning performance and needs to be chosen carefully.

B. Tractable Reformulation

At first glance, the $\inf - \sup$ problem (8) appears to be intractable since the inner supremum has no closed form. However, if we define $\boldsymbol{\Theta} \triangleq \mathbf{x}\mathbf{x}^\top \in \mathbb{S}^{d\times d}$ and $\boldsymbol{\Theta}_n \triangleq \mathbf{x}_n\mathbf{x}_n^\top \in \mathbb{S}^{d\times d}$, the smoothness-based graph learning problem (5) can be rewritten as

$$\inf_{\mathbf{L}\in\mathcal{L}} \mathbb{E}_{\boldsymbol{\Theta}\sim\mathbb{Q}_n}\left[\mathrm{Tr}(\mathbf{L}\boldsymbol{\Theta})\right] + r(\mathbf{L}), \tag{10}$$

where

$$\mathbb{Q}_n = \frac{1}{N}\sum_{n=1}^{N}\mathrm{Dirac}(\mathbf{x}_n\mathbf{x}_n^\top) = \frac{1}{N}\sum_{n=1}^{N}\mathrm{Dirac}(\boldsymbol{\Theta}_n). \tag{11}$$

Thus, we can construct the uncertainty set centered at $\mathbb{Q}_n$ instead of $\mathbb{P}_n$, i.e.,

$$\mathcal{Q} = \left\{\mathbb{Q} :\ W_\alpha(\mathbb{Q}, \mathbb{Q}_n) \leq \epsilon\right\}. \tag{12}$$

Problem (8) can be recast in the form of variable $\boldsymbol{\Theta}$

$$\inf_{\mathbf{L}\in\mathcal{L}} R(\mathbf{L};\boldsymbol{\Theta}) + r(\mathbf{L}) = \inf_{\mathbf{L}\in\mathcal{L}}\sup_{\mathbb{Q}\in\mathcal{Q}} \mathbb{E}_{\boldsymbol{\Theta}\sim\mathbb{Q}}\left[\mathrm{Tr}(\mathbf{L}\boldsymbol{\Theta})\right] + r(\mathbf{L}). \tag{13}$$

Then, we have the following theorem.

Theorem 1: The $\inf - \sup$ problem of (13) is equivalent to the following problem

$$\inf_{\mathbf{L}\in\mathcal{L}} R(\mathbf{L};\boldsymbol{\Theta}) + r(\mathbf{L}) = \inf_{\mathbf{L}\in\mathcal{L}} \mathrm{Tr}(\mathbf{L}\widetilde{\boldsymbol{\Theta}}) + r(\mathbf{L}) + \epsilon\|\mathrm{vec}(\mathbf{L})\|_q, \tag{14}$$

where $\widetilde{\mathbf{\Theta}} = \frac{\mathbf{\Theta}_1+\mathbf{\Theta}_2+....\mathbf{\Theta}_N}{N} = \frac{\mathbf{x}_1\mathbf{x}_1^\top+\mathbf{x}_2\mathbf{x}_2^\top+...+\mathbf{x}_N\mathbf{x}_N^\top}{N}$, and $q \geq 1$ is a constant satisfying $\frac{1}{p}+\frac{1}{q}=1$.

Proof: The proof is inspired by Theorem 2.1 in [19], with one distinction being the usage of $r(\mathbf{L})$ in place of $-\log|\mathbf{L}|$, where $|\mathbf{L}|$ means the (pseudo)determinant of $\mathbf{L}$. Furthermore, [19] use the optimal transport cost to measure the distance between two distributions, while ours use Wasserstein distance. Although the two are similar, α does not affect the final result (14) in our problem. The main idea of the proof is to first provide a dual form of the worst-case expected risk $R(\mathbf{L};\mathbf{\Theta})$ via Corollary 2 in [33] and then show that the dual problem of $R(\mathbf{L};\mathbf{\Theta})$ has the same closed-form solution in the two cases of $\alpha=1$ and $\alpha>1$. After calculating the closed-form solution of $R(\mathbf{L};\mathbf{\Theta})$, we plug it into the (13) and then we obtain (14). See [19], [33] for more details. □

Remark 1: It is observed from Theorem 1 that robustness is achieved via a regularization term of ℓ_q norm, and the uncertainty set size controls the weight of the regularization term as well as the level of robustness. Note that when $q=2$, the regularizer is $\|\mathbf{L}\|_\mathrm{F}$, which is similar to the squared Frobenius norm regularizer $\|\mathbf{L}\|_\mathrm{F}^2$ used to control edge sparsity [8]. However, the $\|\mathbf{L}\|_\mathrm{F}^2$ regularizer is empirically designed without any theoretical interpretation. On the other hand, [12] also attempts to interpret the $\|\mathbf{L}\|_\mathrm{F}$ regularizer from the perspective of DRO. Our model considers the more general case of $q \geq 1$. Furthermore, the model in [12] introduces another regularizer $\kappa\|\mathbf{L}^{\frac{1}{2}}\widehat{\boldsymbol{\mu}}_n\|_2$, where $\widehat{\boldsymbol{\mu}}_n$ is the sample mean vector of the observed signals and κ is the weight. The additional regularizer increases the number of parameters to be determined and complicates the problem further.

C. Out-of-Sample Performance

Next, we provide the OOS performance of the proposed graph estimator to illustrate how the regularizer contributes to the robustness of the learned graphs. The analysis is based on the Rademacher complexity [35], a widely-employed tool to measure the complexity of a class of functions. We first make several technical assumptions.

Assumption 1: The data $\mathbf{x}$ is bounded, i.e., there exists a constant B_x such that $\|\mathbf{x}\|_1 \leq B_x$.

Assumption 2: The ℓ_q norm of the vectorized $\mathbf{L} \in \mathcal{L}$ is bounded, i.e., there exists a constant B_L such that $\|\mathrm{vec}(\mathbf{L})\|_q \leq B_L$.

Assumption 1 naturally holds in the real world. Assumption 2 typically holds since any meaningful graph will have finite edge weights. With these assumptions at hand, we have the following Lemma.

Lemma 1: Define a class of functions $\mathcal{F} \triangleq \{\mathbf{x} \mapsto f(\mathbf{x};\mathbf{L}) : f(\mathbf{x};\mathbf{L}) = \mathrm{Tr}(\mathbf{L}\mathbf{\Theta}), \mathbf{\Theta} = \mathbf{x}\mathbf{x}^\top, \mathbf{L} \in \mathcal{L}\}$ and $B_R \triangleq B_L B_x^2$. Under Assumptions 1-2, the empirical Rademacher complexity of $\mathcal{F}$ of N observed data, denoted as $\mathcal{R}_N(\mathcal{F})$, is bounded by

$$\mathcal{R}_N(\mathcal{F}) \leq \frac{2B_R}{\sqrt{N}}. \tag{15}$$

Proof: See Appendix A. □

Using Lemma 1, we obtain the following theorem.

Theorem 2: Let $\widehat{\mathbf{L}}$ be the solution to (14) using the observed data $\mathbf{x}_1,...,\mathbf{x}_N$. Suppose we draw a new i.i.d. data $\mathbf{x}'$. Under Assumptions 1-2, for any $0<\eta<1$, with probability at least $1-\eta$, we have

$$\mathbb{E}_{\mathbf{x}'\sim\mathbb{P}^*}\left[f(\mathbf{x}';\widehat{\mathbf{L}})\right] \tag{16}$$

$$\leq \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}_n;\widehat{\mathbf{L}}) + \frac{2B_R}{\sqrt{N}} + B_R\sqrt{\frac{8\log(2/\eta)}{N}}, \tag{17}$$

and for any $\xi \geq \frac{2B_R}{\sqrt{N}} + B_R\sqrt{\frac{8\log(2/\eta)}{N}}$,

$$\Pr\left(f(\mathbf{x}';\widehat{\mathbf{L}}) \geq \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}_n;\widehat{\mathbf{L}}) + \xi\right)$$

$$\leq \frac{\frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}_n;\widehat{\mathbf{L}}) + \frac{2B_R}{\sqrt{N}} + B_R\sqrt{\frac{8\log(2/\eta)}{N}}}{\frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}_n;\widehat{\mathbf{L}}) + \xi}. \tag{18}$$

Proof: See Appendix B. □

Given the graph $\widehat{\mathbf{L}}$ learned from (14), the theorem provides an upper bound on the expected loss of smoothness for a new data $\mathbf{x}'$. It says that, with probability at least $1-\eta$, the expected loss of smoothness on the new data is bounded above by the sample average loss of the observed data plus two terms related to the supremum of the regularizer $\|\mathrm{vec}(\mathbf{L})\|_q$, i.e., $B_R = B_L B_x^2$, which corroborates the validity and necessity of our regularization formulation of (14) from the perspective of generalization ability. The two terms related to the WDRO-induced regularizer decay to zero as N increases, which is expected since more observations lead to better OOS performance. Furthermore, (18) reveals that the probability that OOS risk exceeds the sample average risk will also decrease as N increases. When $N \to +\infty$, both (17) and (18) imply that the OOS performance will only depend on the sample average loss of the observed data. However, when N is small, the proposed regularizer contributes bounding the risk of unseen data, promoting the robustness of the graph estimator. Note that Theorem 2 does not provide any insights into how different choices of q for the ℓ_q norm affect the OOS performance. Accordingly, we cannot obtain any suggestion about the choice of q from Theorem 2. Fortunately, we can exploit algorithm unrolling to automatically choose q from data, as we will explain in the next section.

IV. Algorithm Unrolling For Robust Graph Learning

In this section, we first develop an algorithm to iteratively solve the induced optimization problem (14) based on the ADMM framework. We next unroll the proposed algorithm into a neural network to avoid excessive parameter searches.

A. The ADMM Algorithm

We have the following equation according to [7]

$$\mathrm{Tr}(\mathbf{L}\widetilde{\mathbf{\Theta}}) = \frac{1}{N}\mathrm{Tr}(\mathbf{X}^\top\mathbf{L}\mathbf{X}) = \frac{1}{2N}\|\mathbf{W}\circ\mathbf{Z}\|_{1,1}, \tag{19}$$

where $\|\cdot\|_{1,1}$ is the elementwise ℓ_1 norm of a matrix. Besides, $\mathbf{Z}$ is the pairwise distance matrix defined as

$$\mathbf{Z}_{[ij]} = \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2, \tag{20}$$

where $\tilde{\mathbf{x}}_i \in \mathbb{R}^N$ is the i-th row vector of $\mathbf{X}$, i.e., $\mathbf{X} = \left[\tilde{\mathbf{x}}_1^\top, ..., \tilde{\mathbf{x}}_d^\top\right]^\top$. The number of free variables of $\mathbf{W}$ is $m \triangleq \frac{d(d-1)}{2}$, i.e., the upper right triangle elements. Thus, we define two vectors $\mathbf{w}, \mathbf{z} \in \mathbb{R}^m$ whose elements correspond to the upper right entries of $\mathbf{W}$ and $\mathbf{Z}$, respectively. Using (19), the problem (14) can be rephrased in the following vector form

$$\begin{aligned} &\inf_{\mathbf{w}} \frac{\mathbf{z}^\top \mathbf{w}}{N} + \epsilon \left(2\|\mathbf{w}\|_q^q + \|\mathbf{S}\mathbf{w}\|_q^q\right)^{\frac{1}{q}} + \tilde{r}(\mathbf{w}) \\ &\text{s.t. } \mathbf{w} \geq 0,\ \mathbf{w}^\top \mathbf{1} = \frac{d}{2}, \end{aligned} \tag{21}$$

where $\mathbf{S}$ is a linear operator satisfying $\mathbf{S}\mathbf{w} = \mathbf{W}\mathbf{1}$, and $\tilde{r}(\mathbf{w})$ represents $r(\mathbf{L})$ in the form of variable $\mathbf{w}$. Here, $\mathbf{w} \geq 0$ means that all elements of $\mathbf{w}$ are non-negative. By defining $\mathcal{W} \triangleq \left\{\mathbf{w} : \mathbf{w} \geq 0, \mathbf{w}^\top \mathbf{1} = \frac{d}{2}\right\}$, we rewrite (21) as

$$\begin{aligned} &\inf_{\mathbf{w} \in \mathcal{W}} \frac{\mathbf{z}^\top \mathbf{w}}{N} + \epsilon \left(2\|\mathbf{w}\|_q^q + \|\mathbf{S}\mathbf{w}\|_q^q\right)^{\frac{1}{q}} + \tilde{r}(\mathbf{v}) \\ &\text{s.t. } \mathbf{v} = \mathbf{w}. \end{aligned} \tag{22}$$

The scaled augmented Lagrangian form of (22) is

$$\begin{aligned} L_\rho(\mathbf{w}, \mathbf{v}, \mathbf{u}) = &\frac{\mathbf{z}^\top \mathbf{w}}{N} + \epsilon \left(2\|\mathbf{w}\|_q^q + \|\mathbf{S}\mathbf{w}\|_q^q\right)^{\frac{1}{q}} + \tilde{r}(\mathbf{v}) \\ &+ \frac{\rho}{2}\|\mathbf{w} - \mathbf{v} + \mathbf{u}\|_2^2 - \frac{\rho}{2}\|\mathbf{u}\|_2^2, \end{aligned} \tag{23}$$

where $\rho > 0$ is the ADMM penalty parameter, and $\mathbf{u}$ is the dual variable. Under the ADMM framework, the updates of our algorithm are

$$\begin{aligned} \mathbf{w}^{k+1} &= \operatorname*{argmin}_{\mathbf{w} \in \mathcal{W}} L_\rho(\mathbf{w}, \mathbf{v}^k, \mathbf{u}^k) \\ \mathbf{v}^{k+1} &= \operatorname*{argmin}_{\mathbf{v}} L_\rho(\mathbf{w}^{k+1}, \mathbf{v}, \mathbf{u}^k) \\ \mathbf{u}^{k+1} &= \mathbf{u}^k + \mathbf{w}^{k+1} - \mathbf{v}^{k+1}. \end{aligned} \tag{24}$$

The details of the above three updates are as follows.

Update w: The sub-problem of updating $\mathbf{w}$ is

$$\begin{aligned} \mathbf{w}^{k+1} &= \operatorname*{argmin}_{\mathbf{w} \in \mathcal{W}} L_\rho(\mathbf{w}, \mathbf{v}^k, \mathbf{u}^k) \\ &= \operatorname*{argmin}_{\mathbf{w} \in \mathcal{W}} \frac{\mathbf{z}^\top \mathbf{w}}{N} + \epsilon \left(2\|\mathbf{w}\|_q^q + \|\mathbf{S}\mathbf{w}\|_q^q\right)^{\frac{1}{q}} \\ &\quad + \frac{\rho}{2}\|\mathbf{w} - \mathbf{v}^k + \mathbf{u}^k\|_2^2. \end{aligned} \tag{25}$$

We leverage the projected gradient descent algorithm to solve the sub-problem. Specifically, the gradient of the objective function of (25) is

$$\begin{aligned} &\nabla_{\mathbf{w}} L_\rho(\mathbf{w}, \mathbf{v}^k, \mathbf{u}^k) \\ &= \epsilon \left(2\|\mathbf{w}\|_q^q + \|\mathbf{S}\mathbf{w}\|_q^q\right)^{\frac{1-q}{q}} \cdot \left(2\mathbf{w}^{\cdot(q-1)} + \mathbf{S}^\top (\mathbf{S}\mathbf{w})^{\cdot(q-1)}\right) \\ &\quad + \rho\left(\mathbf{w} - \mathbf{v}^k + \mathbf{u}^k\right) + \frac{\mathbf{z}}{N}. \end{aligned} \tag{26}$$

Let $\widetilde{\mathbf{w}}_0^k = \mathbf{w}^k$, and perform the following updates T times

$$\widetilde{\mathbf{w}}_{t+1}^k = \mathrm{Proj}_{\mathcal{W}}\left(\widetilde{\mathbf{w}}_t^k - \nu \nabla_{\mathbf{w}} L_\rho(\widetilde{\mathbf{w}}_t^k, \mathbf{v}^k, \mathbf{u}^k)\right), \tag{27}$$

Algorithm 1 The ADMM Algorithm for (14)

```
Input:
    $\epsilon$, $\rho$, $q$, $\nu$, graph signals $\mathbf{X}$
Output:
    The learned graph $\mathbf{w}$
1:  Initialize $\mathbf{w}^0$, $\mathbf{v}^0$ and $\mathbf{u}^0$, and set $k = 0$
2:  while stop criterion not satisfied do
3:      Let $\widetilde{\mathbf{w}}_0^k = \mathbf{w}^k$
4:      for $t = 0, ..., T-1$ do
5:          $\widetilde{\mathbf{w}}_{t+1}^k = \mathrm{Proj}_{\mathcal{W}}\left(\widetilde{\mathbf{w}}_t^k - \nu \nabla L_\rho(\widetilde{\mathbf{w}}_t^k, \mathbf{v}^k, \mathbf{u}^k)\right)$
6:      end for
7:      Let $\mathbf{w}^{k+1} = \widetilde{\mathbf{w}}_T^k$
8:      $\mathbf{v}^{k+1} = \mathrm{prox}_{\frac{\tilde{r}}{\rho}}\left(\mathbf{w}^{k+1} + \mathbf{u}^k\right)$
9:      $\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{w}^{k+1} - \mathbf{v}^{k+1}$
10:     $k = k + 1$
11: end while
12: return $\mathbf{w}^k$
```

where ν is the (constant) stepsize, and $\mathrm{Proj}_{\mathcal{W}}(\cdot)$ denotes the projection into the simplex $\mathcal{W}$. Finally, we let $\mathbf{w}^{k+1} = \widetilde{\mathbf{w}}_T^k$.

Update v: The sub-problem of updating $\mathbf{v}$ is

$$\begin{aligned} \mathbf{v}^{k+1} &= \operatorname*{argmin}_{\mathbf{v}}\ L_\rho(\mathbf{w}^{k+1}, \mathbf{v}, \mathbf{u}^k) \\ &= \operatorname*{argmin}_{\mathbf{v}}\ \tilde{r}(\mathbf{v}) + \frac{\rho}{2}\|\mathbf{w}^{k+1} - \mathbf{v} + \mathbf{u}^k\|_2^2. \end{aligned} \tag{28}$$

The solution of (28) is

$$\mathbf{v}^{k+1} = \mathrm{prox}_{\frac{\tilde{r}}{\rho}}\left(\mathbf{w}^{k+1} + \mathbf{u}^k\right), \tag{29}$$

where $\mathrm{prox}_{\frac{\tilde{r}}{\rho}}(\cdot)$ is the proximal operator of $\frac{\tilde{r}}{\rho}$.

Update u: The update of $\mathbf{u}$ is straight, i.e., $\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{w}^{k+1} - \mathbf{v}^{k+1}$.

We update $\mathbf{w}$, $\mathbf{v}$ and $\mathbf{u}$ alternately until the solutions converge. The ADMM framework can guarantee global convergence if the problem is convex [36]. Furthermore, the stopping criterion of the ADMM framework is that the primal and dual errors are both smaller than a predefined threshold. See [36] for more details. The complete algorithm flow is shown in Algorithm 1. The computational burden of Algorithm 1 is mainly on $\nabla_{\mathbf{w}} L_\rho(\mathbf{w}, \mathbf{v}, \mathbf{u})$, $\mathrm{Proj}_{\mathcal{W}}(\cdot)$, and $\mathrm{prox}(\cdot)$. The computational cost of $\nabla_{\mathbf{w}} L_\rho(\mathbf{w}, \mathbf{v}, \mathbf{u})$ is $\mathcal{O}(m)$, where $m = \frac{d(d-1)}{2}$ and d is the number of nodes. For commonly used graph structural regularizer like logarithm barrier, the cost is also $\mathcal{O}(m)$. The projection $\mathrm{Proj}_{\mathcal{W}}(\cdot)$ requires $\mathcal{O}(m + \mathrm{nnz}(\mathrm{Proj}_{\mathcal{W}}(\mathbf{w})) \cdot \log m)$ as stated in [12], where $\mathrm{nnz}(\cdot)$ is the number of nonzeros. Thus, the overall complexity is $K_1 T(m + \mathrm{nnz}(\mathrm{Proj}_{\mathcal{W}}(\mathbf{w})) \cdot \log m)$, where K_1 is the number of outer iterations to converge.

We see that in Algorithm 1, we need to manually tune parameters including ϵ, ρ, q and ν. Moreover, $\tilde{r}(\mathbf{w})$ may also have some extra parameters, see [7], [8]. Finding a high-performance combination of them manually is not easy due to the large search space. Furthermore, as mentioned before, the determination of ϵ is crucial since it reflects the level of robustness. As such, it is essential to devise a convenient method to determine all parameters automatically.

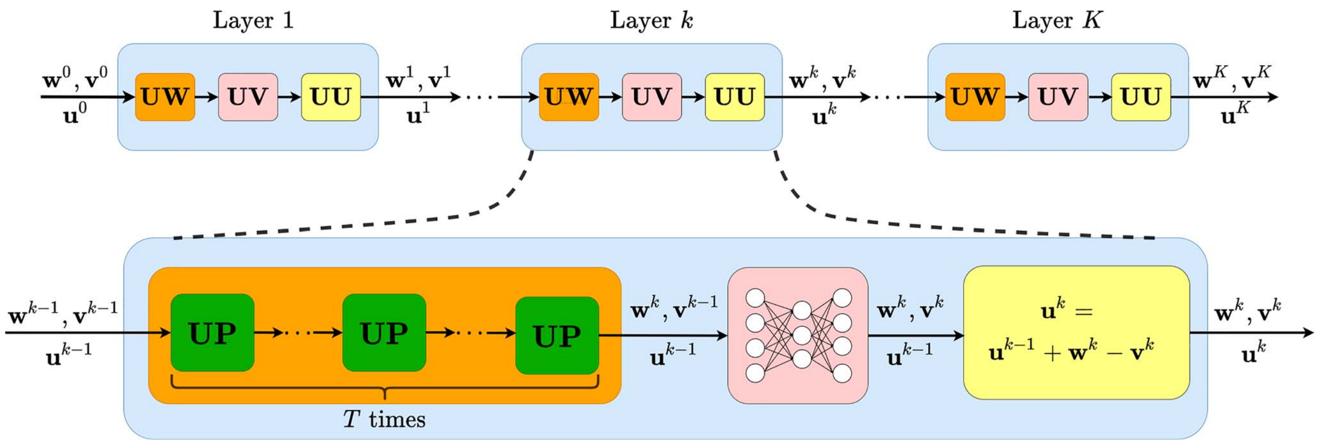

Layer 1
Layer k
Layer K
UW
UV
UU
$\mathbf{w}^0, \mathbf{v}^0$
$\mathbf{u}^0$
$\mathbf{w}^1, \mathbf{v}^1$
$\mathbf{u}^1$
$\mathbf{w}^k, \mathbf{v}^k$
$\mathbf{u}^k$
$\mathbf{w}^K, \mathbf{v}^K$
$\mathbf{u}^K$
$\mathbf{w}^{k-1}, \mathbf{v}^{k-1}$
$\mathbf{u}^{k-1}$
UP
T times
$\mathbf{w}^k, \mathbf{v}^{k-1}$
$\mathbf{u}^{k-1}$
$\mathbf{w}^k, \mathbf{v}^k$
$\mathbf{u}^{k-1}$
$\mathbf{u}^k = \mathbf{u}^{k-1} + \mathbf{w}^k - \mathbf{v}^k$
$\mathbf{w}^k, \mathbf{v}^k$
$\mathbf{u}^k$

Fig. 1. An illustration of the proposed unrolled network.

B. Unrolling The ADMM Algorithm

In this subsection, we unroll the proposed ADMM algorithm into a neural network, in which the parameters to be determined in Algorithm 1 are learned by training the network. The details of the unrolled network are described next.

1) Network Structure: Suppose we run the ADMM algorithm K iterations, and each iteration corresponds to one layer in the unrolled network. As displayed in Fig. 1, the network contains three modules per layer, i.e., **UW**, **UV** and **UU**. We next explain the three modules in detail.

UW: The module **UW** is related to updating $\mathbf{w}$, which consists of T **UP** submodules. In each **UP** submodule, the network makes one forward propagation via (27) to update $\mathbf{w}$. Note that we need to perform the operator $\mathrm{Proj}_{\mathcal{W}}(\cdot)$ in **UP** to project variables into the simplex $\mathcal{W}$. Many efficient algorithms have been developed to implement the projection; see [37] and the references therein. In this study, we develop a new projection method in which all operations can be back-propagated to train the unrolled networks. Specifically, for any vector $\mathbf{s} \in \mathbb{R}^m$, the projected vector $\mathbf{w} \in \mathcal{W}$ is

$$\mathbf{w} = \max\left(\mathbf{s} - \max\left\{\frac{d/2 - \sum_{i=1}^{I} \mathbf{s}^{[I]}}{-I}, I = 1, ...m\right\}, 0\right), \tag{30}$$

where $\max(\cdot)$ is an element-wise operator, $\mathbf{s}^{[I]}$ is the I-th largest element of $\mathbf{s}$. It is not difficult to verify that all operations of (30) can be back-propagated. The proposed method is similar to [38], and the difference is that our method does not need to determine the number of non-zero elements in $\mathbf{s}$. The derivation of (30) is presented in Appendix C.

UV: The module **UV** is related to updating $\mathbf{v}$. In the ADMM algorithm, the update of $\mathbf{v}$ is the proximal operator of a handcrafted function $\tilde{r}$. However, the design of $\tilde{r}$ requires topological priors about the desired graphs, which may not accurately describe the true graph topology. Indeed, the proximal operator can be viewed as a mapping that projects a vector into the space that satisfies some topological properties [9]. A natural question is, can we replace the handcraft mapping with some well-designed plugin, or directly learn the mapping from samples? Some explorations have been made on this problem, and an excellent example is PnP-ADMM [39], [40], [41]. In PnP-ADMM, the mapping is replaced by a denoiser to enhance image restoration performance [41]. Another line of work attempts to leverage a neural network to replace the mapping, such as convolutional neural networks [42]. The recent work [9] designs a variational autoencoder to replace the mapping $\mathbf{w} \geq 0$ when we infer graph topology. In this study, we replace the proximal operator of the regularizer $\tilde{r}(\mathbf{w})$ with a multi-layer neural network $\mathcal{NN}$, which is more intuitive than [9] since $\tilde{r}(\mathbf{w})$ is directly related to the topological priors. The update of this module is written as

$$\mathbf{v}^{k+1} = \mathcal{NN}(\mathbf{w}^{k+1} + \mathbf{u}^k). \tag{31}$$

The structure of $\mathcal{NN}$ is $m \times N_h \times N_h \times m$, where N_h is the size of hidden layers of $\mathcal{NN}$. We let $N_h = 8$ in our model. The activation functions of two hidden layers are ReLU functions.

UU: The module **UU** is related to updating $\mathbf{u}$, i.e., $\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{w}^{k+1} - \mathbf{v}^{k+1}$.

The whole procedure are shown in Algorithm 2. For Algorithm 2, the proximal operator $\mathrm{prox}(\cdot)$ is replaced by a neural network $\mathcal{NN}$. The corresponding cost is $\mathcal{O}(mN_h + N_hN_h + N_hm)$. Thus, the overall cost of Algorithm 2 is $K_2T(m + \mathrm{nnz}(\mathrm{Proj}_{\mathcal{W}}(\mathbf{w})) \cdot \log m + mN_h + N_hN_h + N_hm)$. Note that K_2 is usually much smaller than K_1 thanks to unrolling [9].

2) Trainable Parameters: In the unrolled network, the first class of trainable variables are those in **UW**, i.e., ϵ, ρ, q, and stepsize ν. To enhance the capacity of the unrolled network, we let the parameters of each iteration in **UP** be independent. Therefore, the final trainable parameters are ϵ_t^k, ρ_t^k, q_t^k and ν_t^k for $t = 0, ..., T-1$ and $k = 0, ..., K-1$. The other trainable parameters are those in $\mathcal{NN}$. In summary, the total number of trainable parameters is $4TK + |\mathcal{NN}|$, where $|\mathcal{NN}|$ is the number of parameters of $\mathcal{NN}$.

3) Loss Function: The loss function consists of two parts, i.e., the squared relative error (SRE) and the connection accuracy (CA). The SRE is used to measure the errors of the learned

Algorithm 2 The Unrolled ADMM Algorithm

```
Input:
    Number of layers $K, T$, and signals $\mathbf{X}$
Output:
    The learned graph $\mathbf{w}$
1:  Initialize $\mathbf{w}^0$, $\mathbf{v}^0$ and $\mathbf{u}^0$ and set $k = 0$
2:  for $k = 0, \ldots, K-1$ do
3:      // The module UW
4:      Let $\widetilde{\mathbf{w}}_0^k = \mathbf{w}^k$
5:      for $t = 0, \ldots, T-1$ do
6:          // The submodule UP
7:          $\widetilde{\mathbf{w}}_{t+1}^k = \mathrm{Proj}_{\mathcal{W}}\left(\widetilde{\mathbf{w}}_t^k - \nu_t^k \nabla L_\rho(\widetilde{\mathbf{w}}_t^k, \mathbf{v}^k, \mathbf{u}^k)\right)$
8:      end for
9:      Let $\mathbf{w}^{k+1} = \widetilde{\mathbf{w}}_T^k$
        // The module UV
10:     $\mathbf{v}^{k+1} = \mathcal{NN}(\mathbf{w}^{k+1} + \mathbf{u}^k)$
        // The module UU
11:     $\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{w}^{k+1} - \mathbf{v}^{k+1}$
12:     $k = k + 1$
13: end for
14: return $\mathbf{w}^K$
```

edge weights and is defined as

$$SRE_t^k = \frac{\|\widetilde{\mathbf{w}}_t^k - \mathbf{w}^*\|_2^2}{\|\mathbf{w}^*\|_2^2}, \tag{32}$$

where $\mathbf{w}^*$ is the ground-truth graph. The loss CA is used to measure the accuracy of the learned graph topology. In graph learning, identifying whether there is a connection between two vertices can be viewed as a binary classification problem. Therefore, we employ binary cross-entropy loss as CA

$$CA_t^k = BCE\left(\mathrm{sgn}(\mathbf{w}^*), \Pr(\widetilde{\mathbf{w}}_t^k)\right), \tag{33}$$

where

$$\left(\mathrm{sgn}\left(\mathbf{w}^*\right)\right)_{[i]} = \begin{cases} 1, & \text{if } \mathbf{w}_{[i]}^* > 0 \\ 0 & \text{others.} \end{cases} \tag{34}$$

Here, $\Pr(\widetilde{\mathbf{w}}_t^k)$ is the probability value of $\widetilde{\mathbf{w}}_t^k$ passing through a sigmoid function. The final loss function is

$$Loss = \frac{1}{M}\sum_{j=1}^{M}\sum_{k=0}^{K-1}\sum_{t=1}^{T}\tau^{TK-kT-t}\left(\left(SRE_t^k\right)_j + \zeta\left(CA_t^k\right)_j\right), \tag{35}$$

where ζ is the trade-off weight between SRE and CA, and $\tau \in (0, 1]$ is a loss-discounting factor added to reduce the contribution of the first few layers. Moreover, $(SRE_t^k)_j$ and $(CA_t^k)_j$ represent the SRE and CA of $\widetilde{\mathbf{w}}_t^k$ calculated from the j-th training samples. We use M training samples, $\{(\mathbf{X}_1, \mathbf{w}_1^*), \ldots, (\mathbf{X}_M, \mathbf{w}_M^*)\}$, where $\mathbf{X}_j \in \mathbb{R}^{d\times N}$ contains N observed signals, and $\mathbf{w}_j^*$ is taken as the ground-truth label. Compared to the traditional cross-validation method, AU may need more training samples to obtain the optimal parameters. However, compared to traditional neural networks, the unrolled network incorporates domain knowledge about graph learning models and hence requires fewer training data to extract information. The advantage of AU over cross-validation is that it provides a fine-grained approach to automatically learn parameters in an end-to-end manner, thereby exhibiting better performance.

4) Network Training: Stacking all the modules mentioned above, we train the unrolled network end-to-end by minimizing (35). The number of layers K, T and M are set to 2, 10 and 3000 respectively. For the loss function, we set $\tau = 0.9$ and $\zeta = 1$. At the beginning of the training process, the parameters ϵ_t^k, q_t^k, ρ_t^k and ν_t^k are initialized as $1, 2, 0.5$ and 0.05, respectively. During training, we restrict the parameters to the feasible region. The parameters of $\mathcal{NN}$ are initialized randomly. The maximum training epochs is set to 100. We select Adam as our optimizer and set a decaying learning rate.

V. Experiments

In this section, we test the performance of the proposed framework using both synthetic data and real-world data. First of all, some experimental setups are introduced.

A. Experimental Setups

1) Graph Generation: We generate three types of random graphs, i.e., Gaussian radial basis function (RBF) graphs [8], Erdős–Rényi (ER) graphs [43], and stochastic block model (SBM) graphs [44]. We follow the same approach as [8], including parameter settings, to generate the RBF graphs. The ER and SBM graphs are generated using the python package *networkx*. The probability that two nodes are connected in ER graphs is 0.2. For SBM graphs, the inter-block connection probability is set to 0.15, while the inner-block connection probability is in $[0.8, 0.95]$.

2) Graph Signals Generation: We generate graph signals from the following distribution

$$\mathbf{x} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{L}^\dagger + \sigma_w^2 \mathbf{I}\right), \tag{36}$$

where $\mathbf{L}$ is the Laplacian matrix of $\mathcal{G}$, and σ_w denotes the noise level. As described in [8], graph signals generated in this way are smooth over the corresponding graphs.

3) Synthetic Dataset Preparation: After obtaining M graphs $\mathbf{w}_j^*, j = 1, \ldots, M$, based on the graph generation methods, we generate N signals $\mathbf{X}_j$ based on (36) for each graph. Then, the training dataset containing M data is constructed as $\{(\mathbf{X}_1, \mathbf{w}_1^*), \ldots, (\mathbf{X}_M, \mathbf{w}_M^*)\}$ as mentioned before.

4) Evaluation Metric: For topology inference, identifying whether two nodes are connected is a binary classification problem. Thus, we first leverage Precision, Recall and Matthews correlation coefficient (MCC) [45] to evaluate the classification results, which are defined as follows

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}, \quad \mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}},$$

$$\mathrm{MCC} = \frac{\mathrm{TP}\cdot\mathrm{TN} - \mathrm{FP}\cdot\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}, \tag{37}$$

where TP is the true positive rate, TN is the true negative rate, FP is the false positive rate, and FN denotes the false negative rate. MCC has been argued to be one of the most informative coefficients for assessing the performance of binary classification since it summarizes all information from TP, TN, FP and FN. The value range of MCC is $[-1, +1]$, where $+1$ and -1 represent completely correct and wrong identification. Some studies have employed MCC to evaluate the learned graphs, such as [19]. The second type of metric is the relative error (RE) of edge weights between the learned graphs and the ground-truth, i.e.,

$$\mathrm{RE} = \frac{\|\widehat{\mathbf{w}} - \mathbf{w}^*\|_2}{\|\mathbf{w}^*\|_2}, \tag{38}$$

where $\widehat{\mathbf{w}}$ is the learned graphs, and $\mathbf{w}^*$ is the ground-truth. The final results are the average of 50 independent experiments.

5) Baselines: We employ four classes of baseline models. The first category includes traditional smoothness-based graph learning models using iterative algorithms, i.e., the SigRep model [8], the LogDet model [46], and the Log model [7]. The second category contains two graph learning models using AU, i.e., GLAD [31] and L2G [9]. Note that the graphs learned from GLAD are represented by precision matrices, but we still use it as a baseline to test the effect of unrolling different iterative algorithms. The third class of model, MUGL [12], is the only smoothness-based graph learning model we can find in the literature that considers data uncertainty. Finally, we let $\widetilde{r}(\mathbf{w})$ be the logarithmic barrier in [7] and use the Algorithm 1 to solve the problem, which is termed LogADMM. The parameters of all iterative algorithms are determined by grid search as those corresponding to the optimal MCC values. The parameters of GLAD are the same as those in [31]. For a fair comparison, the L2G model has 20 layers and a maximum epoch of 100.

B. Synthetic Data

1) Model Performance: We investigate the performance of our framework under different levels of data uncertainty. Four cases are considered, where N is set to 500 and 3000, and σ_w is set to 0.1 and 0.8, respectively. The results are displayed in Table I, from which it is observed that our model achieves the best performance (MCC and RE) in almost all cases. Furthermore, the standard deviations of our model are relatively small, indicating that the learned graphs are consistent across different observed data. The MUGL model also obtains small standard deviations since it considers moment uncertainty in the data distributions. However, it fails to achieve competitive MCC and RE values. Besides, the uncertainty set size is determined manually in MUGL, which may not match the real situation. The performance of GLAD is sensitive to noise levels. It fails to learn meaningful graphs when $\sigma_w = 0.8$ and bears large standard deviations when $\sigma_w = 0.1$. Another AU model, the L2G, obtains satisfactory RE values. For example, when $N = 500$ and $\sigma_w = 0.8$, the RE of L2G for RBF graphs is smaller than ours. However, the topology recovery MCC of L2G is inferior to ours, and L2G tends to learn dense graphs in our experiments. This may be caused by the fact that the loss function of L2G does not consider the connection accuracy of the learned graphs. Moreover, the smooth signals are

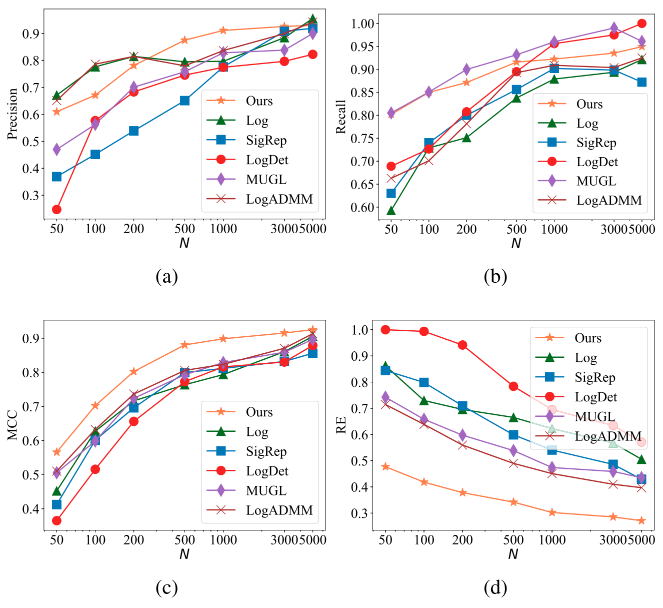


Fig. 2. The impact of varying N.

generated from $\mathcal{N}\left(\mathbf{0}, (\mathbf{L} + \sigma_w^2 \mathbf{I})^{\dagger}\right)$ in [9], while ours are from (36). Note that our method is a standard method for generating smooth signals from a signal representation perspective, and has been widely used in graph learning problems such as [8], [12]. Finally, the performance of LogADMM is inferior to ours for all metrics. The reasons may be two-fold. First, all parameters of LogADMM such as q are manually selected. The combinations of these parameters may deviate from the optimal ones. Our unrolling algorithm can automatically learn appropriate parameters from data, thus showing better results. Second, instead of the predetermined logarithmic barrier prior, we leverage a neural network $\mathcal{NN}$ to learn topological priors from data, which could better capture the graph structure behind the data.

Table II lists the learned q and ϵ by training the unrolled networks. The results are the average of all T **UP** submodules of K layers. We can conclude from the results that for more data uncertainty (smaller N and larger σ_w), a large ϵ will be learned. This makes sense because a large uncertainty set is required to account for more out-of-sample cases if the empirical distribution is unreliable. Furthermore, it is observed that the learned q is around 2.0 and is difficult to determine using priors. Fortunately, we provide a data-driven method to automatically select the optimal q.

We also test the performance of our method on varying N. Fig. 2 depicts the results of the case where $\sigma_w = 0.5$ and the underlying graphs are generated based on the ER model. It is observed that our method obtains the best MCC and Recall over a wide range of N. When N is large, the performance of baselines is close to ours since the data uncertainty decreases as N. Our method may not achieve the best Precision and RE for some N, but it can achieve the best MCC, which is a more comprehensive metric for evaluating topological inference results.

TABLE I
THE PERFORMANCE ON SYNTHETIC DATA UNDER DIFFERENT UNCERTAINTY LEVELS

		$N=500,\ \sigma_w=0.1$				$N=500,\ \sigma_w=0.8$			
		Precison	Recall	MCC	RE	Precision	Recall	MCC	RE
RBF	Log	**0.969±9.04%**	0.795±5.16%	0.812±10.31%	0.669±6.13%	**0.972±5.62%**	0.599±6.54%	0.674±8.03%	0.760±6.33%
	SigRep	0.784±9.97%	0.915±8.81%	0.803±13.09%	0.369±5.60%	0.690±8.23%	0.989±4.30%	0.694±9.67%	0.400±5.26%
	LogDet	0.936±8.64%	0.865±16.21%	0.842±14.63%	0.575±11.17%	0.901±8.38%	0.866±13.97%	0.811±11.32%	0.761±7.63%
	MUGL	0.819±8.87%	0.946±2.88%	0.817±10.82%	0.458±6.48%	0.824±7.11%	0.945±2.64%	0.802±7.78%	0.509±5.48%
	GLAD	0.906±9.92%	0.742±12.68%	0.735±11.92%	0.691±16.68%	—	—	—	—
	L2G	0.453±5.98%	**0.994±2.38%**	0.341±9.13%	0.327±4.49%	0.432±5.59%	**0.997±1.65%**	0.263±15.44%	**0.355±8.01%**
	LogADMM	0.875±7.61%	0.901±6.25%	0.822±10.31%	0.344±6.02%	0.792±6.25%	0.918±4.81%	0.814±7.52%	0.386±7.01%
	Ours	0.937±3.91%	0.937±3.06%	**0.899±4.08%**	**0.311±7.30%**	0.896±5.90%	0.926±4.13%	**0.855±6.98%**	0.370±6.74%
ER	Log	**0.959±8.86%**	0.885±4.52%	0.841±9.51%	0.561±12.28%	0.839±9.90%	0.731±5.68%	0.760±8.37%	0.687±4.52%
	SigRep	0.819±7.88%	0.937±4.69%	0.809±6.03%	0.555±3.85%	0.580±3.10%	0.975±6.30%	0.792±4.90%	0.571±3.17%
	LogDet	0.903±10.32%	0.893±5.33%	0.840±10.10%	0.670±15.10%	0.723±15.67%	0.904±9.00%	0.761±9.07%	0.827±5.03%
	MUGL	0.878±6.81%	0.976±3.17%	0.833±5.22%	0.503±7.95%	0.591±6.53%	0.941±3.68%	0.786±4.79%	0.544±6.17%
	GLAD	0.897±8.11%	0.735±10.43%	0.731±10.13%	0.708±11.54%	—	—	—	—
	L2G	0.295±4.43%	**0.994±1.24%**	0.321±8.12%	0.449±6.76%	0.267±4.93%	**0.997±1.18%**	0.269±10.99%	0.496±6.91%
	LogADMM	0.883±7.18%	0.932±4.35%	0.851±5.54%	0.473±6.825%	0.772±6.18%	0.894±6.81%	0.802±6.78%	0.520±7.12%
	Ours	0.907±6.13%	0.939±4.06%	**0.902±4.78%**	**0.336±8.23%**	**0.843±8.01%**	0.891±5.30%	**0.829±6.52%**	**0.432±8.56%**
SBM	Log	0.947±3.76%	0.852±5.42%	0.858±4.07%	0.599±4.56%	**0.947±3.92%**	0.685±5.26%	0.739±5.12%	0.714±4.34%
	SigRep	0.831±5.69%	0.922±8.87%	0.838±8.59%	0.464±3.09%	0.714±5.82%	0.945±8.18%	0.817±8.24%	0.495±3.67%
	LogDet	**0.966±3.29%**	0.652±18.65%	0.728±13.46%	0.809±9.04%	0.917±6.05%	0.652±19.33%	0.696±13.06%	0.897±4.77%
	MUGL	0.842±5.31%	0.919±6.78%	0.847±6.20%	0.453±3.65%	0.816±5.97%	0.911±6.60%	0.821±6.48%	0.516±4.79%
	GLAD	0.912±8.31%	0.759±11.02%	0.751±9.84%	0.565±12.96%	—	—	—	—
	L2G	0.362±3.53%	**0.997±1.02%**	0.368±4.03%	0.384±3.17%	0.343±2.93%	**0.992±1.26%**	0.327±6.58%	0.436±3.65%
	LogADMM	0.882±4.79%	0.914±7.27%	0.887±6.19%	0.464±4.53%	0.802±6.04%	0.922±7.08%	0.819±7.21%	0.489±4.87%
	Ours	0.931±3.16%	0. 930±3.42%	**0.902±3.14%**	**0.345±5.91%**	0.856±4.95%	0.902±3.91%	**0.825±5.49%**	**0.427±5.74%**
		$N=3000,\ \sigma_w=0.1$				$N=3000,\ \sigma_w=0.8$			
		Precison	Recall	MCC	RE	Precision	Recall	MCC	RE
RBF	Log	0.987±5.17%	0.834±3.88%	0.854±5.54%	0.588±8.50%	**0.989±5.96%**	0.702±4.69%	0.762±6.73%	0.680±3.66%
	SigRep	**0.989±4.27%**	0.820±4.09%	0.847±4.89%	0.652±7.70%	0.984±5.74%	0.806±3.97%	0.836±6.64%	0.695±6.71%
	LogDet	0.962±6.99%	0.910±7.37%	0.902±8.31%	0.498±15.31%	0.913±10.55%	0.931±7.69%	0.870±9.68%	0.720±7.84%
	MUGL	0.830±6.81%	0.962±2.54%	0.832±7.41%	0.437±6.90%	0.838±6.27%	0.960±2.24%	0.826±6.88%	0.433±6.17%
	GLAD	0.977±8.21%	0.833±15.31%	0.842±14.69%	0.447±25.95%	—	—	—	—
	L2G	0.518±6.77%	**0.964±3.99%**	0.436±7.88%	0.321±15.52%	0.460±6.49%	**0.997±1.68%**	0.336±11.54%	0.414±5.23%
	LogADMM	0.923±4.81%	0.911±3.97%	0.887±5.21%	0.332±7.86%	0.892±5.21%	0.914±3.78%	0.875±6.11%	0.366±7.81%
	Ours	0.956±4.49%	0.943±3.29%	**0.920±5.10%**	**0.262±7.74%**	0.949±3.64%	0.942±2.93%	**0.913±4.15%**	**0.289±8.31%**
ER	Log	0.878±11.54%	0.902±4.97%	0.864±10.96%	0.536±15.54%	**0.973±4.04%**	0.839±9.97%	0.843±8.72%	0.636±4.59%
	SigRep	0.825±10.02%	0.912±4.41%	0.815±9.14%	0.544±13.31%	0.881±3.98%	0.906±8.65%	0.806±8.41%	0.556±10.45%
	LogDet	0.839±10.47%	**0.984±5.87%**	0.874±10.33%	0.635±29.22%	0.749±13.43%	0.980±5.84%	0.810±12.92%	0.794±7.97%
	MUGL	0.828±5.28%	0.952±3.54%	0.852±4.97%	0.495±8.19%	0.823±4.90%	0.990±2.34%	0.844±4.64%	0.495±6.68%
	GLAD	0.821±10.09%	0.934±4.59%	0.838±10.19%	0.471±15.84%	—	—	—	—
	L2G	0.335±5.62%	0.981±2.03%	0.392±6.95%	0.434±9.29%	0.296±3.73%	**0.996±1.06%**	0.330±5.26%	0.445±6.03%
	LogADMM	0.881±8.84%	0.916±4.87%	0.874±8.23%	0.404±10.31%	0.877±6.16%	0.903±5.45%	0.849±7.11%	0.436±9.41%
	Ours	**0.925±7.72%**	0.945±4.30%	**0.918±6.37%**	**0.275±9.65%**	0.874±11.44%	0.951±4.99%	**0.887±9.72%**	**0.286±9.03%**
SBM	Log	0.969±2.20%	0.889±4.46%	0.900±3.61%	0.524±4.24%	**0.984±1.66%**	0.781±5.08%	0.835±3.82%	0.623±2.74%
	SigRep	0.941±2.91%	0.898±4.14%	0.883±3.04%	0.588±5.42%	0.936±3.57%	0.877±4.35%	0.868±3.87%	0.614±5.31%
	LogDet	**0.971±3.83%**	0.845±13.99%	0.866±7.97%	0.709±11.60%	0.954±3.67%	0.825±14.81%	0.842±10.28%	0.850±6.43%
	MUGL	0.942±3.31%	0.887±6.31%	0.880±5.49%	0.410±3.45%	0.946±3.97%	0.869±6.60%	0.865±5.83%	0.419±3.90%
	GLAD	0.953±7.31%	0.892±9.32%	0.891±9.18%	0.402±10.92%	—	—	—	—
	L2G	0.455±9.62%	**0.989±1.68%**	0.496±11.30%	0.365±4.98%	0.366±4.31%	**0.999±0.12%**	0.386±4.89%	0.295±4.13%
	LogADMM	0.937±5.11%	0.908±4.05%	0.895±4.55%	0.398±6.22%	0.921±4.71%	0.892±4.05%	0.877±5.09%	0.421±6.58%
	Ours	0.948±5.09%	0.959±3.09%	**0.934±4.64%**	**0.268±6.14%**	0.945±4.98%	0.951±3.30%	**0.926±4.81%**	**0.289±6.42%**

2) The Impact of Network Structures: Subsequently, we fix $N = 500$ and $\sigma_w = 0.8$, and train the unrolled networks on ER graphs with different K and T. We use a tuple (K, T) to denote different combinations of K and T. As depicted in Fig. 3, compared with the case $(2, 5)$, the performance of $(2, 10)$ improves as T. However, when K and T continue to increase, the improvement is insignificant but brings more computational costs. Thus, we select $(K, T) = (2, 10)$ in previous experiments, which provides satisfactory performance with a low computational burden. Note that our algorithm can achieve competitive performance using a few network layers since the domain knowledge about the smoothness model is incorporated into the unrolled networks and brings more efficient parameters, which is a merit of the AU framework.

2) Ablation Study: To further illustrate the superiority of our model, we conduct some ablation studies. We consider five cases, including (i) fixing $\epsilon = 2$ (FixEps), (ii) fixing $q = 2$ (Fixq), (iii) replacing the $\mathcal{NN}$ module with the logarithmic

TABLE II
THE LEARNED q AND ϵ OF DIFFERENT UNCERTAINTY LEVELS

		$N = 500$ $\sigma_w = 0.1$	$N = 500$ $\sigma_w = 0.8$	$N = 3000$ $\sigma_w = 0.1$	$N = 3000$ $\sigma_w = 0.8$
RBF	q	2.027	2.018	1.864	1.981
	ϵ	9.719	10.604	9.512	10.007
ER	q	2.013	2.263	1.981	1.933
	ϵ	10.659	10.928	9.722	10.123
SBM	q	1.831	1.993	1.863	2.027
	ϵ	9.156	10.717	7.701	9.801

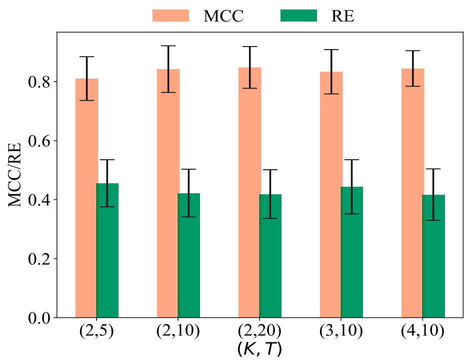


Fig. 3. The impact of the number of the network layers.

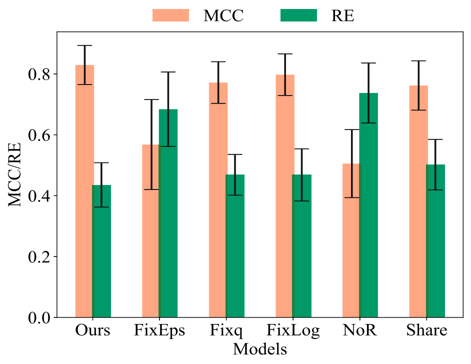


Fig. 4. The results of different ablation studies.

barrier in [7] (FixLog), (iv) removing the regularizer related to robustness and keeping the hyperparameters for the network unchanged (NoR), and (v) making different layers of the unrolled network share the same trainable parameters (Share). We employ ER graphs and set $N = 500, \sigma_w = 0.8$ in this experiment. As displayed in Fig. 4, our model outperforms the cases (i) - (ii). This can be explained by the fact that the manually selected ϵ and q may be inconsistent with the optimal one. Our model directly learns the optimal q and ϵ from the data, thus exhibiting better performance. The first two cases demonstrate the advantage of learnable parameters. Furthermore, our model also outperforms the FixLog because it learns the latent topological properties through the module $\mathcal{NN}$. In contrast, the FixLog model utilizes a hand-crafted function to impart some prior properties to the learned graphs. The NoR model achieves the worst performance on both MCC and RE and displays higher

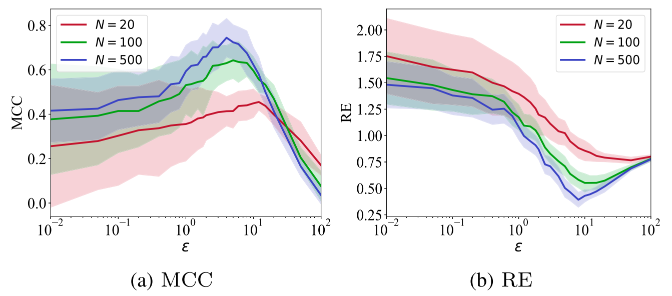
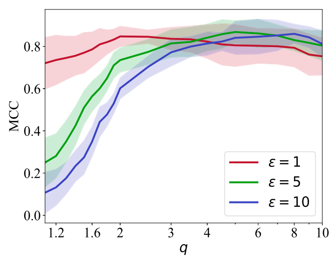


Fig. 5. The performance of different ϵ. The shaded area is the interval covered by standard deviations.

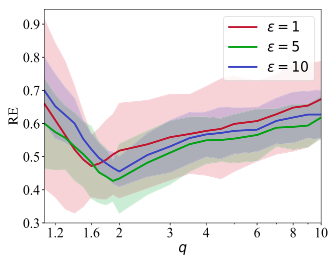
standard deviations as it ignores the data uncertainty. This case verifies the necessity of the robustness-related regularizer. Finally, our model yields better performance than the case Share. The reason may be that our model has a larger network capacity (each layer has its individual parameters), and thus performs better.

3) The Impact of ϵ: Although the unrolled network can learn the optimal ϵ automatically, we still study the impact of ϵ to verify the plausibility of the proposed robust graph learning model. In this experiment, we learn ER graphs using Algorithm 1—instead of the unrolled network—with different ϵ. Besides, we fix $q = 2, \sigma_w = 0.1, \rho = 1, \nu = 0.001$ and choose the logarithmic barrier function [7] as $\widetilde{r}(\mathbf{w})$. We remark that these parameters may not be optimal since we only focus on the effect of ϵ rather than finding the optimal model. Fig. 5 displays the performance of the learned ER graphs for different ϵ. Two trends can be observed. (i) As ϵ increases, the performance first improves and then degrades. The optimal ϵ decreases as the increase of N. This can be interpreted as, given the observed data, there exists an ϵ that best matches the data uncertainty. For a larger N, the observed data contains less uncertainty, meaning that a small ϵ is sufficient to contain the true distribution. When ϵ is too large, the performance deteriorates since the uncertainty set contains too many nuisance distributions, making the learned graph over-conservative. Note that the optimal values of ϵ in this experiment differ from those of our unrolled network since the network learns all optimal parameters simultaneously. In contrast, we fix the other parameters to be constant in this experiment. (ii) The performance fluctuation (standard deviations) decreases as ϵ increases because the worst-case expected risk of large ϵ will consider more OOS distributions, resulting in more consistent performance. The performance of different ϵ behaves as expected, which justifies our model.

4) The Impact of q: In addition to ϵ, we also investigate the impact of q. Specifically, we let $N = 500, \sigma_w = 0.1, \rho = 1$, and $\nu = 0.001$. We select graph structural regularizer as the logarithmic barrier function [7] and learn ER graphs using Algorithm 1. As illustrated in Fig. 5, for a given ϵ, there exists an optimal q such that MCC and RE reach the optimal value. The larger ϵ is, the larger the optimal q value is. However, it is difficult to build a quantitative relationship between ϵ and the optimal q. In addition, there are other factors that may affect the optimal q value, such as the choice of the graph

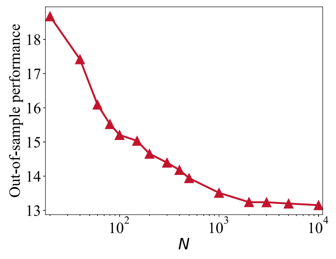


(a) MCC

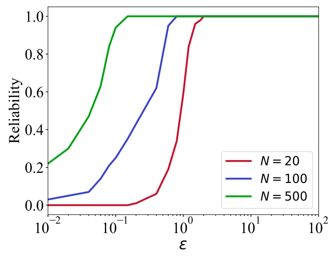


(b) RE

Fig. 6. The performance of different q. The shaded area is the interval covered by standard deviations.



(a) Out-of-sample performance



(b) Reliability

Fig. 7. The (a) OOS performance and (b) reliability of the learned graphs.

structural regularizer. Fortunately, our unrolling algorithm can directly learn the optimal combination of q and ϵ in a data-driven manner, as shown in Table II.

5) Out-of-Sample Performance: Finally, we investigate the OOS performance of our proposed model. We fix $q = 2$, $\sigma_w = 0.1$, $\rho = 1$, $\nu = 0.001$, and ϵ is selected via cross-validation. We generate $N_u = 10^4$ signals from ER graphs as the unobserved data and calculate the sample average risk (SAR) of the unseen data, $\mathrm{SAR} = \frac{1}{N_u}\sum_{n=1}^{N_u} f(\mathbf{x}'_n; \widehat{\mathbf{L}})$, as the OOS performance, where $\widehat{\mathbf{L}}$ is the graph learned from our model using the observed data. Fig. 7(a) shows that the OOS performance first improves (SAR decreases) rapidly and then levels off as N increases. As illustrated in Theorem 2, if N is large enough, the terms related to the WDRO-induced regularizer in (17)-(18) will decay to zero. The OOS performance then only depends on the SAR of the observed data, which becomes almost stable for large N. Then, we investigate the OOS performance of our model from a new angle by defining

$$\mathrm{Reliability} = \Pr\left(f(\mathbf{x}'; \widehat{\mathbf{L}}) \le R^*\right), \tag{39}$$

where R^* is the obtained worst-case risk by solving (14) using the observed data. The Reliability represents the probability that the OOS risk is smaller than the worst-case risk, which can also be interpreted as the probability that the uncertainty set contains the ground-truth distribution. To calculate Reliability, we run 100 independent simulations. We calculate the SAR of 10^4 unobserved data in each simulation. We employ the ratio of $\mathrm{SAR} \le R^*$ in 100 independent simulations as Reliability. Fig. 7(b) depicts that as ϵ increases, the Reliability approaches

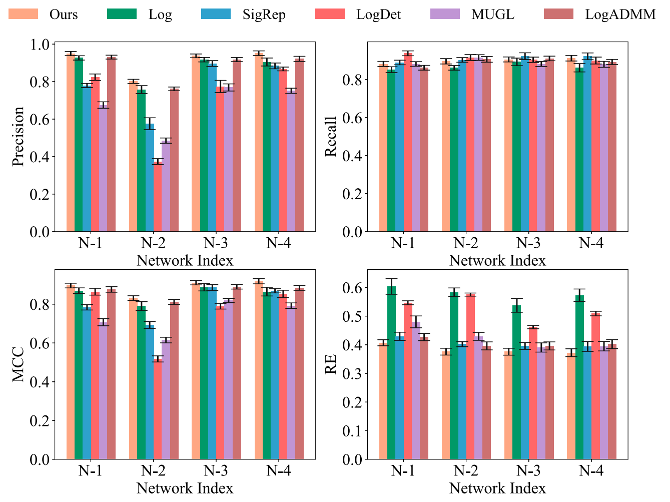


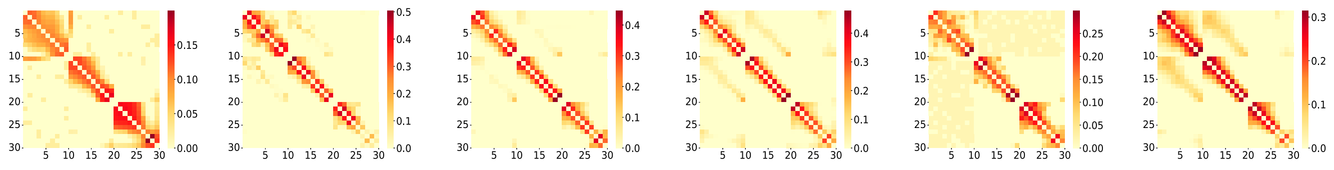
Fig. 8. The performance of the learned social networks. N-1 to N-4 represent the selected networks.

1, indicating that the OOS risk is smaller than R^* with high confidence. However, high Reliability is not equivalent to learning good graphs since the R^* of large ϵ may also be worse. The reason is that the large uncertainty set may take into account too many nuisance distributions, resulting in the learned graphs being over-conservative, as shown in Fig. 5.

C. Real-World Data

1) Social Networks: We first consider the social networks from the Ljubljana student social network dataset[1]. The dataset contains 12 networks whose nodes represent the same 32 students. Edges in these networks capture the interactions among these students, which are built on the answers of these students to different questions. Each student is asked 12 questions corresponding to 12 networks. Note that the dataset does not contain graph signals. Thus, we randomly select four networks and generate signals for each network using (36). The reason we employ this "semi-real" dataset is that we hope to evaluate the learned graphs quantitatively. We generate 1000 signals with $\sigma_w = 0.5$ for each network. The learned graphs are evaluated using Precision, Recall, MCC and RE since we have ground-truth graphs. We follow [9] and use a network pre-trained on ER graphs. The networks is trained with $N = 1000$ and $\sigma_w = 0.5$ and then transferred to learn the social networks. As shown in Fig. 8, the baselines can obtain comparable or even better Recall performance than our method. However, our method outperforms the baselines in terms of Precision performance, meaning that our method can recover the social networks more accurately. For both MCC and RE, our model obtains the best performance, indicating the superiority of our method. The Log model achieves the worst RE performance, while the LogDet is sensitive to different graphs. The results of MUGL are not as satisfactory as expected, which may be caused by the inappropriate uncertainty set sizes. Finally, LogADMM is inferior to ours since the parameter search is coarse-grained and

[1]http://vladowiki.fmf.uni-lj.si/doku.php?id=pajek:data:pajek:students



(a) Ours (b) Log (c) SigRep (d) LogDet (e) MUGL (f) LogADMM

Fig. 9. The visualizations of the graphs learned by different models.

TABLE III
THE DETECTION RESULTS OF THE LEARNED GRAPHS

	Ours	Log	SigRep	LogDet	MUGL	LogADMM
NMI	1.000	0.834	0.830	0.698	0.836	0.907
FMI	1.000	0.727	0.714	0.507	0.733	0.864
RI	1.000	0.811	0.802	0.643	0.816	0.906

* The larger the three indicators, the more accurate.

the hand-craft regularizer may not accurately describe the real graph properties. The standard deviations in this experiment are smaller than those of previous experiments since the ground-truth graphs remain unchanged.

2) The COIL-20 Dataset: We next leverage the COIL-20[2] dataset, which is a collection of gray-scale images of 20 objects taken from 360 degrees. The size of each image is 32×32, and each object has 72 images (five degrees an image). We randomly select 30 images classified into three objects, i.e., ten images per object. We treat each image as a node and aim to learn a relationship graph between these images. The image itself is taken as graph signals, i.e., $\mathbf{X} \in \mathbb{R}^{30\times 1024}$. It is expected that the learned graph should have three communities because the selected images belong to three objects, and the images of the same object are similar. We utilize the SBM graphs as the graph generator since the SBM tends to generate graphs with communities, which is suitable for this dataset. We then use [47] to estimate the noise levels of the images in COIL-20 and obtain $\sigma_w = 1.05$, which is the average of all the estimated noise levels of all images. We then generate 1024 signals from the SBM graphs with $\sigma_w = 1.05$, which are used to train our unrolled network. The trained network is used to learn the relationship graphs of images. To evaluate the learned graphs, we use the Louvain method [48] to detect the communities in the learned graphs. Three commonly used metrics, i.e., normalized mutual information (NMI), Fowlkes and Mallows index (FMI) [49], and Rand Index (RI) are adopted to evaluate the detection results. The labels of the images are taken as the ground-truth, and the detection results are listed in Table III. Our model obtains the highest NMI, FMI, and RI, meaning it can better learn the cluster structures in the graphs. Fig. 9 displays the learned graphs of different models. The graph learned by our model clearly shows three clusters. Compared with graphs learned by other models, there are fewer confusing edges between images of different items, while images of the same item are more closely related. Therefore, our model can capture the topological features of the graphs, such as cluster structures.

[2]https://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php

VI. CONCLUSION

In this paper, we proposed a framework to infer graph topology from smooth signals by considering data uncertainty. Specifically, inspired by the WDRO framework, we formulated the robust graph learning as an $\inf - \sup$ problem and then reformulated it into a tractable form, where robustness is achieved via a regularizer. To confirm the necessity of the regularizer, we conducted an OOS performance analysis of our model. The results indicate that the regularizer can improve the generalization of the learned graphs by bounding the OOS risks with high probability. Then, we proposed an ADMM algorithm to solve the induced optimization problem and further unrolled it into a neural network to avoid excessive parameter searches. The parameters of our framework, including the uncertainty set size, are determined automatically by training the network. Extensive synthetic and real data experiments showed that our model outperforms state-of-the-art methods. Future work will include incorporating our framework into the unrolled graph signal denoising model [29], [30], in which graphs are fixed. Developing a more scalable method is another possible direction.

APPENDIX A
PROOF OF LEMMA 1

At the beginning of the proof, we first prove that $|f(\mathbf{x};\mathbf{L})| \leq B_L B_x^2$. Specifically, we have

$$\begin{aligned} |f(\mathbf{x};\mathbf{L})| &= |\mathrm{Tr}(\mathbf{L}\boldsymbol{\Theta})| \\ &\leq \|\mathrm{vec}(\mathbf{L})\|_q \|\mathrm{vec}(\boldsymbol{\Theta})\|_p \\ &\leq \|\mathrm{vec}(\mathbf{L})\|_q \|\mathrm{vec}(\boldsymbol{\Theta})\|_1 \\ &\leq B_L B_x^2, \end{aligned} \tag{40}$$

where the first inequality holds due to Hölder inequality. The second inequality holds due to the theorem of norm equivalence [50]. The last inequality holds since $\|\mathrm{vec}(\boldsymbol{\Theta})\|_1 = \|\mathbf{x}\|_1^2 \leq B_x^2$. Next, by following [51], suppose that $\delta_1, ..., \delta_N$ are i.i.d. uniform random variables on $\{-1, 1\}$. Based on the definition of Rademacher complexity, we have

$$\begin{aligned} \mathcal{R}_N(\mathcal{F}) &= \mathbb{E}\left[\sup_{f\in\mathcal{F}} \frac{2}{N}\left|\sum_{n=1}^{N} \delta_n f(\mathbf{x}_n;\mathbf{L})\right|\right] \\ &\leq \frac{2\beta B_L B_x^2}{N}\mathbb{E}\left[\left|\sum_{n=1}^{N}\delta_n\right|\right] \\ &\leq \frac{2\beta B_L B_x^2}{N}\mathbb{E}\left[\sqrt{\sum_{n=1}^{N}\delta_n^2}\right] \end{aligned}$$

$$= \frac{2\beta B_L B_x^2}{\sqrt{N}} = \frac{2B_R}{\sqrt{N}}, \tag{41}$$

where the first inequality holds due to (40).

Appendix B
Proof of Theorem 2

The proof is derived from Theorem 8 in [35]. However, we need to make some modifications. First, we need to write our problem in a supervised form since Theorem 8 in [35] applies to supervised problems, whereas ours is unsupervised. Specifically, we define $g(\mathbf{x}, y; \mathbf{L}) = f(\mathbf{x}; \mathbf{L}) - 0$ as the new loss function of smoothness, where the "label" y is always equal to zero. Accordingly, the input space $\mathcal{X}$, action space $\mathcal{A}$, and output space $\mathcal{Y}$ in Theorem 8 in [35] correspond to $\mathbb{R}^d$, $\mathbb{R}_+$, and $\{0\}$, respectively. Then, we set the following symbol correspondences: $\mathcal{L}(\mathbf{x}, y) = \phi(\mathbf{x}, y) = g(\mathbf{x}, y; \mathbf{L}) = f(\mathbf{x}; \mathbf{L}) = \mathrm{Tr}(\mathbf{L}\boldsymbol{\Theta})$, where $\mathcal{L}$ and ϕ denote loss functions in [35]. For illustrative purposes, we still use the same notations as [35] in this study despite the abuse of symbols. The second modification is that the value range of the loss function in [35] is set to be $[0, 1]$, while we have $|f(\mathbf{x}; \mathbf{L})| \le B_R$ in our model. With the above two modifications, we can directly yield (17). Next, we apply Markov's inequality and obtain

$$\begin{aligned}
&\Pr\left(f(\mathbf{x}'; \widehat{\mathbf{L}}) \ge \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}_n; \widehat{\mathbf{L}}) + \xi\right) \\
&\le \frac{\mathbb{E}_{\mathbf{x}' \sim \mathbb{P}^*}\left[f(\mathbf{x}'; \widehat{\mathbf{L}})\right]}{\frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}_n; \widehat{\mathbf{L}}) + \xi} \\
&\le \frac{\frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}_n; \widehat{\mathbf{L}}) + \frac{2B_R}{\sqrt{N}} + B_R\sqrt{\frac{8\log(2/\eta)}{N}}}{\frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}_n; \widehat{\mathbf{L}}) + \xi}.
\end{aligned} \tag{42}$$

Appendix C
Derivation of Projection Operator (30)

For a vector $\mathbf{s} \in \mathbb{R}^m$, we aim to project it into the region

$$\mathcal{W} = \left\{\mathbf{w} : \mathbf{w} \ge 0, \mathbf{1}^\top \mathbf{w} = d/2\right\}, \tag{43}$$

which is equivalent to solving the following problem

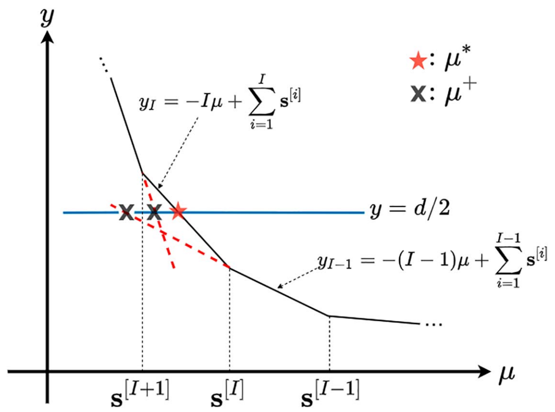
$$\begin{aligned}
&\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w} - \mathbf{s}\|_2^2 \\
&\text{s.t. } \mathbf{w} \ge 0,\ \mathbf{1}^\top \mathbf{w} = d/2.
\end{aligned} \tag{44}$$

The Lagrangian form of (44) can be written as

$$L = \frac{1}{2}\|\mathbf{w} - \mathbf{s}\|_2^2 + \mu\left(\mathbf{1}^\top \mathbf{w} - d/2\right) - \boldsymbol{\lambda}^\top \mathbf{w}, \tag{45}$$

where $\mu \in \mathbb{R}$ and $\boldsymbol{\lambda} \in \mathbb{R}^m$ are the Lagrangian multipliers. The KKT conditions are then

$$\begin{aligned}
&\mathbf{w}^* \ge 0, \quad \boldsymbol{\lambda}^* \ge 0, \quad \mathbf{1}^\top \mathbf{w}^* = \frac{d}{2}, \\
&\boldsymbol{\lambda}^*_{[i]} \mathbf{w}^*_{[i]} = 0, \mathbf{w}^*_{[i]} - \mathbf{s}_{[i]} + \mu^* - \boldsymbol{\lambda}^*_{[i]} = 0, \ \text{ for } i = 1, \dots, m,
\end{aligned} \tag{46}$$



y
$y_I = -I\mu + \sum_{i=1}^{I} \mathbf{s}^{[i]}$
★: μ^*
X: μ^+
$y = d/2$
$y_{I-1} = -(I-1)\mu + \sum_{i=1}^{I-1} \mathbf{s}^{[i]}$
μ
$\mathbf{s}^{[I+1]}$
$\mathbf{s}^{[I]}$
$\mathbf{s}^{[I-1]}$

Fig. 10. Illustration of $y(\mu)$. The μ^* is the largest of all possible μ_I^+.

where the variables with superscript $*$ denote the optimal values to be sought. Based on the KKT conditions, it is not difficult to conclude that $\mathbf{w}^*_{[i]}, i = 1, \dots, m$, should satisfy

$$\mathbf{w}^*_{[i]} = \begin{cases} \mathbf{s}_{[i]} - \mu^*, & \mu^* < \mathbf{s}_{[i]} \\ 0, & \mu^* \ge \mathbf{s}_{[i]}. \end{cases} \tag{47}$$

Plugging (47) back into the constraint, $\mathbf{1}^\top \mathbf{w} = \frac{d}{2}$, we have

$$\sum_{i=1}^{m} \max\left\{\mathbf{s}_{[i]} - \mu^*, 0\right\} = d/2. \tag{48}$$

We then define a function

$$y(\mu) = \sum_{i=1}^{m} \max\left\{\mathbf{s}_{[i]} - \mu, 0\right\}. \tag{49}$$

Observe that $y(\mu)$ is a piecewise function as shown in Fig. 10. We need to find the μ^* such that $y(\mu^*) = d/2$. We sort all entries in $\mathbf{s}$ in descending order and denote $\mathbf{s}^{[i]}$ as the i-th largest element. The I-th segment of the function is

$$y_I(\mu) = -I\mu + \sum_{i=1}^{I} \mathbf{s}^{[i]}. \tag{50}$$

We denote μ_I^+ as the solution of the equation $y_I(\mu) = d/2$. Thanks to the convexity of $y(\mu)$, the solution of $y(\mu) = d/2$ is the largest of all μ_I^+, i.e.,

$$\mu^* = \max\left\{\frac{d/2 - \sum_{i=1}^{I} \mathbf{s}^{[i]}}{-I}, \ I = 1, \dots, m\right\}. \tag{51}$$

It is not difficult to obtain the $\mathbf{w}^*$ as

$$\mathbf{w}^* = \max(\mathbf{s} - \mu^*, 0). \tag{52}$$

Plugging (51) into (52), we complete the proof.

References

[1] D. Thanou, X. Dong, D. Kressner, and P. Frossard, "Learning heat diffusion graphs," *IEEE Trans. Signal. Inf. Process. Netw.*, vol. 3, no. 3, pp. 484–499, Mar. 2017.

[2] U. Von Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007.

[3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2020.

[4] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, "Connecting the dots: Identifying network structure via graph signal processing," *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 16–43, Mar. 2019.

[5] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 44–63, Mar. 2019.

[6] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, May 2018.

[7] V. Kalofolias, "How to learn a graph from smooth signals," in *Proc. Int. Conf. Artif. Intell. Stat., AISTATS*. PMLR, 2016, pp. 920–929.

[8] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Learning laplacian matrix in smooth graph signal representations," *IEEE Trans. Signal Process.*, vol. 64, no. 23, pp. 6160–6173, 2016.

[9] X. Pu, T. Cao, X. Zhang, X. Dong, and S. Chen, "Learning to learn graph topologies," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, pp. 4249–4262, 2021.

[10] P. Berger, G. Hannak, and G. Matz, "Efficient graph learning from noisy and incomplete data," *IEEE Trans. Signal. Inf. Process. Netw.*, vol. 6, pp. 105–119, 2020.

[11] S. P. Chepuri, S. Liu, G. Leus, and A. O. Hero, "Learning sparse graphs under smoothness prior," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.* Piscataway, NJ, USA: IEEE Press, 2017, pp. 6508–6512.

[12] X. Wang, Y.-M. Pun, and A. M.-C. So, "Distributionally robust graph learning from smooth signals under moment uncertainty," *IEEE Trans. Signal Process.*, vol. 70, pp. 6216–6231, 2022.

[13] H. Rahimian and S. Mehrotra, "Distributionally robust optimization: A review," 2019, *arXiv:1908.05659*.

[14] A. Ben-Tal, D. Den Hertog, A. De Waegenaere, B. Melenberg, and G. Rennen, "Robust solutions of optimization problems affected by uncertain probabilities," *Manage. Sci.*, vol. 59, no. 2, pp. 341–357, 2013.

[15] E. Delage and Y. Ye, "Distributionally robust optimization under moment uncertainty with application to data-driven problems," *Oper. Res.*, vol. 58, no. 3, pp. 595–612, 2010.

[16] P. M. Esfahani and D. Kuhn, "Data-driven distributionally robust optimization using the Wasserstein metric: Performance guarantees and tractable reformulations," *Math. Program.*, vol. 171, no. 1, pp. 115–166, 2018.

[17] S. S. Abadeh, V. A. Nguyen, D. Kuhn, and P. M. M. Esfahani, "Wasserstein distributionally robust Kalman filtering," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.

[18] V. A. Nguyen, D. Kuhn, and P. M. Esfahani, "Distributionally robust inverse covariance estimation: The Wasserstein shrinkage estimator," 2018, *arXiv:1805.07194*.

[19] P. Cisneros-Velarde, A. Petersen, and S.-Y. Oh, "Distributionally robust formulation and model selection for the graphical lasso," in *Proc. Int. Conf. Artif. Intell. Stat., AISTATS*. PMLR, 2020, pp. 756–765.

[20] P. Mohajerin Esfahani and D. Kuhn, "Data-driven distributionally robust optimization using the Wasserstein metric: Performance guarantees and tractable reformulations," *Math. Program.*, vol. 171, no. 1-2, pp. 115–166, 2018.

[21] D. Kuhn, P. M. Esfahani, V. A. Nguyen, and S. Shafieezadeh-Abadeh, "Wasserstein distributionally robust optimization: Theory and applications in machine learning," in *Operations Res. Manage. Sci. Age Analytics*. INFORMS, 2019, pp. 130–166.

[22] S. Shafieezadeh-Abadeh, D. Kuhn, and P. M. Esfahani, "Regularization via mass transportation," *J. Mach. Learn. Res.*, vol. 20, no. 103, pp. 1–68, 2019.

[23] J. Blanchet, Y. Kang, and K. Murthy, "Robust Wasserstein profile inference and applications to machine learning," *J. Appl. Probab.*, vol. 56, no. 3, pp. 830–857, 2019.

[24] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Process. Mag.*, vol. 38, no. 2, pp. 18–44, Feb. 2021.

[25] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 399–406.

[26] Y. Yang, J. Sun, H. Li, and Z. Xu, "ADMM-CSNET: A deep learning approach for image compressive sensing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 3, pp. 521–538, 2018.

[27] Y. Li, M. Tofighi, V. Monga, and Y. C. Eldar, "An algorithm unrolling approach to deep image deblurring," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.* Piscataway, NJ, USA: IEEE Press, 2019, pp. 7675–7679.

[28] N. Shlezinger, Y. C. Eldar, and S. P. Boyd, "Model-based deep learning: On the intersection of deep learning and optimization," *IEEE Access*, vol. 10, pp. 115384–115398, 2022.

[29] S. Chen, Y. C. Eldar, and L. Zhao, "Graph unrolling networks: Interpretable neural networks for graph signal denoising," *IEEE Trans. Signal Process.*, vol. 69, pp. 3699–3713, 2021.

[30] M. Nagahama, K. Yamada, Y. Tanaka, S. H. Chan, and Y. C. Eldar, "Graph signal restoration using nested deep algorithm unrolling," *IEEE Trans. Signal Process.*, vol. 70, pp. 3296–3311, 2022.

[31] H. Shrivastava et al., "Glad: Learning sparse graph recovery," in *Proc. Int. Conf. Learn. Representations*, 2020.

[32] M. Wasserman, S. Sihag, G. Mateos, and A. Ribeiro, "Learning graph structure from convolutional mixtures," 2022, *arXiv:2205.09575*.

[33] R. Gao and A. J. Kleywegt, "Distributionally robust stochastic optimization with Wasserstein distance," 2016, *arXiv:1604.02199*.

[34] L. Stanković, M. Daković, and E. Sejdić, "Introduction to graph signal processing," in *Vertex-Freq. Anal. Graph Signals*. New York, NY, USA: Springer, 2019, pp. 3–108.

[35] P. L. Bartlett and S. Mendelson, "Rademacher and Gaussian complexities: Risk bounds and structural results," *J. Mach. Learn. Res.*, vol. 3, no. 11, pp. 463–482, 2002.

[36] S. Boyd, N. Parikh, and E. Chu, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Boston, MA, USA: Now Publishers Inc., 2011.

[37] L. Condat, "Fast projection onto the simplex and the l 1 ball," *Math. Program.*, vol. 158, no. 1–2, pp. 575–585, 2016.

[38] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, "Efficient projections onto the l 1-ball for learning in high dimensions," in *Proc. Int. Conf. Mach. Learn.*, 2008, pp. 272–279.

[39] Y. Dar, A. M. Bruckstein, M. Elad, and R. Giryes, "Postprocessing of compressed images via sequential denoising," *IEEE Trans. Image Process.*, vol. 25, no. 7, pp. 3044–3058, Jul. 2016.

[40] S. Ono, "Primal-dual plug-and-play image restoration," *IEEE Signal Process. Lett.*, vol. 24, no. 8, pp. 1108–1112, Aug. 2017.

[41] Y. Yazaki, Y. Tanaka, and S. H. Chan, "Interpolation and denoising of graph signals using plug-and-play ADMM," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process*. Piscataway, NJ, USA: IEEE Press, 2019, pp. 5431–5435.

[42] S. Lohit, D. Liu, H. Mansour, and P. T. Boufounos, "Unrolled projected gradient descent for multi-spectral image fusion," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process*. Piscataway, NJ, USA: IEEE Press, 2019, pp. 7725–7729.

[43] E. N. Gilbert, "Random graphs," *Ann. Math. Statist.*, vol. 30, no. 4, pp. 1141–1144, 1959.

[44] B. Karrer and M. E. Newman, "Stochastic blockmodels and community structure in networks," *Phys. Rev. E*, vol. 83, no. 1, p. 016107, 2011.

[45] D. M. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation," 2020, *arXiv:2010.16061*.

[46] B. Lake and J. Tenenbaum, "Discovering structure by learning sparse graphs," in *Cognit. Sci. Soc.*, vol. 32, no. 32, 2010.

[47] X. Liu, M. Tanaka, and M. Okutomi, "Single-image noise level estimation for blind denoising," *IEEE Trans. Image Process.*, vol. 22, no. 12, pp. 5226–5237, Dec. 2013.

[48] S. Fortunato, "Community detection in graphs," *Phys. Rep.*, vol. 486, no. 3-5, pp. 75–174, 2010.

[49] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*. Amsterdam, The Netherlands: Elsevier, 2011.

[50] M. Goldberg, "Equivalence constants for LP norms of matrices," *Linear Multilinear Algebra*, vol. 21, no. 2, pp. 173–179, 1987.

[51] D. Bertsimas, V. Gupta, and I. C. Paschalidis, "Data-driven estimation in equilibrium using inverse optimization," *Math. Program.*, vol. 153, pp. 595–633, 2015.