LEARNING PRODUCT GRAPHS FROM MULTIDOMAIN SIGNALS
Sai Kiran Kadambari and Sundeep Prabhakar Chepuri
Indian Institute of Science, Bangalore, India

ABSTRACT
In this paper, we focus on learning the underlying product graph structure from multidomain training data. We assume that the product graph is formed from a Cartesian graph product of two smaller factor graphs. We then pose the product graph learning problem as the factor graph Laplacian matrix estimation problem. To estimate the factor graph Laplacian matrices, we assume that the data is smooth with respect to the underlying product graph. When the training data is noise free or complete, learning factor graphs can be formulated as a convex optimization problem, which has an explicit solution based on the water-filling algorithm. The developed framework is illustrated using numerical experiments on synthetic data as well as real data related to air quality monitoring in India.

Index Terms— Graph learning, graph signal processing, Laplacian matrix estimation, product graphs, graph learning, topology inference.

1. INTRODUCTION
Leveraging the underlying structure in data is central to many machine learning and signal processing tasks [1–4]. In many cases, data resides on irregular (non-Euclidean) domains. Some examples include datasets from meteorological stations, traffic networks, social and biological networks, to name a few.

Graphs offer a natural way to describe the structure and explain complex relationships in such network datasets. More specifically, data (signal) is indexed by the nodes of a graph and the edges encode the relationship between the function values at the nodes. When the number of nodes in the graph is very large, signal processing or machine learning operations defined on graphs require more memory and computational resources, e.g., computing the graph Fourier transform [2] using an eigendecomposition requires \( O(N^3) \) operations for a graph with \( N \) nodes [5]. Whenever the underlying graph can be factorized into two or more factor graphs with fewer nodes, the computational costs of these operations can be reduced significantly [1]. Product graphs can efficiently represent multidomain data. For instance, in brain imaging, the data is multidomain as each spatially distributed sensor gathers temporal data. Similarly, in movie recommender systems (such as Netflix) the rating data matrix is a degree vector \( \mathbf{d} \) represents the Kronecker sum and \( \otimes \) represents the Kronecker product.

2. PRODUCT GRAPH SIGNALS
Consider a graph \( G_N = (V_N, \mathcal{E}_N) \) with \( N \) vertices (or nodes), where \( V_N \) denotes the set of vertices and \( \mathcal{E}_N \) denotes the edge set. The structure of the graph with \( N \) nodes is captured by the weighted adjacency matrix \( \mathbf{W} \in \mathbb{R}^{N \times N} \) whose \((i,j)\)th entry denotes the weight of the edge between node \( i \) and node \( j \). When there is no edge between node \( i \) and node \( j \), the \((i,j)\)th entry of \( \mathbf{W} \) is zero. We assume that the graph is undirected with positive edge weights. The corresponding graph Laplacian matrix is a symmetric matrix of size \( N \), given by \( \mathbf{L}_N = \text{diag}([\mathbf{d}]) - \mathbf{W} \), where \( \mathbf{d} \in \mathbb{R}^N \) is a degree vector given by \( \mathbf{d} = \mathbf{W}\mathbf{1} \).

Consider two graphs \( G_P = (V_P, \mathcal{E}_P) \) and \( G_Q = (V_Q, \mathcal{E}_Q) \) with \( P \) and \( Q \) nodes, respectively. Let the corresponding graph Laplacian matrices be \( \mathbf{L}_P \in \mathbb{R}^{P \times P} \) and \( \mathbf{L}_Q \in \mathbb{R}^{Q \times Q} \). Let the Cartesian product of the two graphs \( G_P \) and \( G_Q \) be denoted by \( G_N \) with \( |V_N| = |V_P||V_Q| \) nodes, i.e., \( |E_N| = |E_P||E_Q| \). In other words, the graphs \( G_P \) and \( G_Q \) are the factors of \( G_N \). Then the graph Laplacian matrix \( \mathbf{L}_N \) can be expressed in terms of \( \mathbf{L}_P \) and \( \mathbf{L}_Q \) as

\[
\mathbf{L}_N = \mathbf{L}_P \oplus \mathbf{L}_Q = \mathbf{I}_Q \otimes \mathbf{L}_P + \mathbf{L}_Q \otimes \mathbf{I}_P.
\]
Let us collect the set of graph signals \( \{x_i\}_{i=1}^T \) with \( x_i \in \mathbb{R}^N \) defined on the product graph \( G_N \) in an \( N \times T \) matrix \( X = [x_1, x_2, \ldots, x_T] \). As each node in the product graph \( G_N \) is related to a pair of vertices in its graph factors, we can reshape any product graph signal \( x_i \) as \( x_i \in \mathbb{R}^{P \times Q} \), \( i = 1, 2, \ldots, T \), such that \( x_i = \text{vec}(X_i) \). This means that each product graph signal represents a multidomain signal where the columns and rows of \( X \) are graph signals associated to the graph factor \( G_P \) and \( G_Q \), respectively.

In this work, we will assume that \( X \) is smooth with respect to (w.r.t.) the graph \( G_N \). The amount of smoothness is quantified by the Laplacian quadratic form \( \text{tr}(X^T L_N X) \), where small values of \( \text{tr}(X^T L_N X) \) imply that the data \( X \) is smooth on the graph \( G_N \).

### 3. TASK-COGNIZANT PRODUCT GRAPH LEARNING

Suppose we are given the training data \( X \in \mathbb{R}^{N \times T} \) defined on the product graph \( G_N \), where each column of \( X \), i.e., \( x_i \in \mathbb{R}^N \), represents a multidomain graph data \( X_i \in \mathbb{R}^{P \times Q} \). Assuming that the given data is smooth on the graph \( G_N \) and the product graph \( G_N \) can be factorized as the Cartesian product of two graphs \( G_P \) and \( G_Q \) as in (1), we are interested in estimating the graph Laplacian matrices \( L_P \) and \( L_Q \). To do so, we assume that \( P \) and \( Q \) are known.

Typically, we might not have access to the original graph data \( X \), but we might observe data related to \( X \). Let us call this data \( Y \in \mathbb{R}^{N \times T} \). For example, \( Y \) could be a noisy or an incomplete version of \( X \). Given \( Y \), the joint estimation of \( X \), \( L_P \) and \( L_Q \), may be mathematically formulated as the following optimization problem

\[
\begin{align*}
\min_{L_P \in L_P, L_Q \in L_Q} & \quad f(X, Y) + \alpha \text{tr}(X^T (L_P \oplus L_Q) X) \\
& + \beta_1 \|L_P\|_F^2 + \beta_2 \|L_Q\|_F^2,
\end{align*}
\] (2)

where \( L_N := \{L \in \mathbb{R}^{N \times N} | L1 = 0, \text{tr}(L) = N, L_{ij} = L_{ji} \leq 0, i \neq j \} \) is the space of all the valid Laplacian matrices of size \( N \).

We use the trace equality constraint in this set to avoid a trivial solution. The loss function \( f(X, Y) \) is appropriately chosen depending on the nature of the observed data. More importantly, we learn the product graph suitable for the task of minimizing \( f(X, Y) \). For instance, if the observed graphs signals are noisy as \( y_i = x_i + n_i \) for \( 1 \leq i \leq T \) with the noise vector \( n_i \), then \( f(X, Y) \) is chosen as \( \|X - Y\|_F^2 \). A smoothness-promoting quadratic term is added to the objective function with a positive regularizer \( \alpha \). The squared Frobenius norm of the Laplacian matrices with tuning parameters \( \beta_1 \) and \( \beta_2 \) in the objective function controls the distribution of edge weights. By varying \( \alpha, \beta_1 \), and \( \beta_2 \), we can control the sparsity (i.e., the number of zeros) of \( L_P \) and \( L_Q \), while setting \( \beta_1 = \beta_2 = 0 \) gives the sparest graph.

### 4. SOLVER BY ADAPTING EXISTING WORKS

Ignoring the structure of \( G_N \) that it can be factorized as \( G_P \) and \( G_Q \), and given \( X \) (or its noisy version), one can learn the graph Laplacian matrix \( L_N \) using any one of the methods in [6–9]. In this section, we will develop a solver for the optimization problem (2) based on the existing method in [6].

There exists several numerical approaches for factorizing product graphs, i.e., to find the graph factors \( L_P \) and \( L_Q \) from \( L_N \) [12, 13]. One of the simplest and efficient ways to obtain the graph factors is by solving the convex optimization problem

\[
\begin{align*}
\min_{L_P \in L_P, L_Q \in L_Q} & \quad \|L_N - L_P \oplus L_Q\|_F^2.
\end{align*}
\] (3)

The above problem can be solved using any one of the off-the-shelf convex optimization toolboxes. However, notice that this is a two-step approach, which requires computing a size-\( N \) Laplacian matrix in the first step using [6], for instance. The first step is computationally expensive as the number of nodes in the product graph \( G_N \) increases exponentially with the increase in the number of nodes in either of the factor graphs \( G_P \) and \( G_Q \). In other words, finding the product graph Laplacian matrix and then factorizing it is computationally much expensive and sometimes infeasible because of the huge memory requirement for storing the data. Therefore, in the next section, we propose a computationally efficient solution.

### 5. PROPOSED SOLVER

In this section, instead of a two-step approach, as discussed in Section 4, we provide a one-step solution for estimating the Laplacian matrices of the factors of the product graph by leveraging the properties of the Cartesian graph product and the structure of the data.

The quadratic smoothness promoting term \( \text{tr}(X^T (L_P \oplus L_Q) X) \) can be written as

\[
\begin{align*}
\text{tr}(X^T (L_P \oplus L_Q) X) &= \sum_{i=1}^{T} x_i^T (L_P \oplus L_Q) x_i,
\end{align*}
\] (4)

Using the property of the Cartesian product that for any given matrices \( A, B, C \) and \( X \) of appropriate dimensions, the equation \( AX + XB = C \) is equivalent to \( (A \oplus B)^T \text{vec}(X) = \text{vec}(C) \). Using the property that \( \text{tr}(A^T X) = \text{vec}(A)^T \text{vec}(X) \), (4) simplifies to

\[
\sum_{i=1}^{T} x_i^T (L_P \oplus L_Q) x_i = \sum_{i=1}^{T} \text{vec}(x_i)^T \text{vec}(L_P x_i + x_i L_Q).
\]

This means that the amount of smoothness of the multidomain signal \( x \), w.r.t. \( G_N \) is equal to the sum of the amount of smoothness of the signals collected in the rows and columns of \( X \), w.r.t. \( G_P \) and \( G_Q \), respectively.

Assuming that the training data is noise free and complete, that is, \( Y = X \), and \( X \) available, (2) simplifies to

\[
\begin{align*}
\min_{L_P \in L_P, L_Q \in L_Q} & \quad \alpha \sum_{i=1}^{T} \left[ \text{tr}(X_i^T L_P X_i) + \text{tr}(X_i L_Q X_i^T) \right] \\
& + \beta_1 \|L_P\|_F^2 + \beta_2 \|L_Q\|_F^2.
\end{align*}
\] (5)

with variables \( L_P \) and \( L_Q \). The optimization problem (5) has a unique minimizer as it is convex in \( L_P \) and \( L_Q \). In fact, it can be expressed as a quadratic program (QP) with fewer variables as compared to (5) by exploiting the fact that \( L_P \) and \( L_Q \) are symmetric matrices. In essence, we need to solve for only the lower or upper triangular elements of \( L_P \) and \( L_Q \).

Let the vectorized form of the lower triangular elements of \( L_P \) and \( L_Q \) be denoted by \( \text{vec}(L_P) \in \mathbb{R}^{P(P+1)/2} \) and \( \text{vec}(L_Q) \in \mathbb{R}^{Q(Q+1)/2} \), respectively. Furthermore, \( \text{vec}(L_P) \) and \( \text{vec}(L_Q) \), may be, respectively, related to \( \text{vec}(L_P) \) and \( \text{vec}(L_Q) \) as \( L_P = \text{Avec}(L_P) \), \( L_Q = \text{Bvec}(L_Q) \) using matrices \( A \) and \( B \) of appropriate dimensions. Now, using the fact that \( \text{tr}(X_i^T L_P X_i) = \text{vec}(X_i^T)^T \text{vec}(L_P) + \text{tr}(X_i L_Q X_i^T) = \text{vec}(X_i^T)^T \text{vec}(L_Q) \) and the properties of Frobenius norm, we may rewrite (5) as

\[
\begin{align*}
\min_{z \in \mathbb{R}^R} & \quad \frac{1}{2} z^T P z + q^T z \quad \text{subject to} \quad Cz = d, \quad z \geq 0,
\end{align*}
\] (6)
where \( z = \left[ \text{vec}^T(L_P), \text{vec}^T(L_Q) \right]^T \) is the optimization variable of length \( K = 0.5(P^2 + Q^2 + P + Q) \). Here, \( P = \text{diag}[2\beta_1 A^T A, 2\beta_2 B^T B] \) is the diagonal matrix of size \( K \) and \( q_i = \alpha \sum_{i=1}^{T} q_i \) with \( q_i = \left[ \text{vec}(X_i X_i^T), \text{vec}(X_i^T X_i) \right]^T B \) \( \in \mathbb{R}^K \). The matrix \( C \in \mathbb{R}^{(P+Q+2) \times K} \) and the vector \( d \in \mathbb{R}^{P+Q+2} \) are defined appropriately to represent the trace equality constraints (in the constraint sets) in (5). The problem (6) is a QP in its standard form and can be solved optimally using any one of the off-the-shelf solvers such as CVX [14]. To obtain the graph factors \( L_P \) and \( L_Q \) using the solver based on the existing methods as described in Section 4 requires solving a QP with \( N(N + 1)/2 \) variables for \( L_N \) [6], and subsequently, factorizing \( L_N \) as \( L_P \) and \( L_Q \) as in (3), which requires solving one more QP with \( K \) variables. In contrast, the proposed method requires solving only one QP with \( K \) variables. Thus, the proposed method is computationally efficient as compared to the solver based on the existing methods.

6. WATER-FILLING SOLUTION

In this section, we derive an explicit solution for the optimization problem (6) based on the well-known water-filling approach. By writing the KKT conditions and solving for \( z_i \) leads to an explicit solution.

The Lagrangian function for (6) is given by

\[
\mathcal{L}(z, \lambda, \mu) = \frac{1}{2} z^T P z + q_i^T z + \mu^T (d - Cz) - \lambda^T z,
\]

where \( \lambda \in \mathbb{R}^K \) and \( \mu \in \mathbb{R}^{P+Q+2} \) are the Lagrange multipliers corresponding to the inequality and equality constraints and recall that \( P \) is a diagonal matrix. Then the KKT conditions are given by

\[
P_i z_i^* + q_i - c_i \mu^* - \lambda_i^* = 0, \sum_{i=1}^{N} c_i z_i^* = d, \quad z_i^* \geq 0, \quad \lambda_i^* \geq 0, \quad \lambda_i^* z_i^* = 0,
\]

where \( c_i \) is the \( i \)th column of \( C \). We can now solve the KKT conditions to find \( z_i^* \), \( \lambda_i^* \) and \( \mu^* \). To do so, we eliminate the slack variable \( \lambda_i^* \) and solve for \( z_i^* \). This results in

\[
z_i^* = \max \left\{ 0, P_{i,i}^{-1}(c_i^T \mu^* - q_i) \right\}.
\]

To find \( \mu^* \), we may use \( z_i^* \) in the second KKT condition to obtain

\[
\sum_{i=1}^{N} c_i, \max\{0, P_{i,i}^{-1}(c_i^T \mu^* - q_i)\} = d. \quad \text{(Since this is a linear function in } \mu^*, \text{ we can compute } \mu^*_i \text{ using a simple iterative method. This approach is similar to the water-filling algorithm with multiple water levels [15].}

7. NUMERICAL RESULTS

In this section, we will evaluate the performance of the proposed water-filling based solver (henceforth, referred to as Solver 1) and compare it with the solver based on the existing methods described in Section 4 (henceforth, referred to as Solver 2) on synthetic and real datasets.

7.1. Results on synthetic data

To evaluate the performance of the proposed method, we generate synthetic data on a known graph, which can be factorized. We will use these graph factors as a reference (i.e., ground truth) and compare the estimated graph factors with the ground truth in terms of

| Method | \( L_P \) & \( L_Q \) & \( L_N \) |
|--------|--------|--------|
| Solver 1 | 0.9615 | 0.9841 | 0.9755 |
| Solver 2 | 0.7556 | 0.7842 | 0.7612 |

Table 1: F-measure of the proposed solver (Solver 1) and the solver based on the existing methods (Solver 2).

F-measure, which is a measure of the percentage of correctly recovered edges [6]. Specifically, we generate a graph with \( N = 150 \) nodes, which is formed by the Cartesian product of two community graphs with \( P = 10 \) and \( Q = 15 \) nodes, respectively.

We generate \( T = 50 \) signals \( x_i \) that are smooth w.r.t. \( G_N \) by using the factor analysis model described in [6]. Given \( X \), we learn the graph factors using Solver 1 (using the water-filling method) and Solver 2 (wherein we learn the complete graph Laplacian matrix of \( N = 150 \) nodes as in [6] and then factorize it by solving (3)).

In Table 1, we show the best (in terms of the tuning parameters) F-measure scores obtained by Solver 1 and Solver 2. The F-measure of Solver 1 is close to 0.98, which means that the proposed method learns the graph factors close to the ground truth. In Fig. 1, we plot the Laplacian matrices of the ground truth, learned Laplacian matrices from Solver 1 and Solver 2. We see that the graph factors estimated from Solver 1 are more consistent with the ground truth than the factors estimated using Solver 2. Moreover, we stress the fact that the proposed water-filling method is computationally efficient as compared to Solver 2.

7.2. Joint matrix completion and learning graph factors

We now test the performance on real data. For this purpose, we use the PM2.5 data collected over 40 air quality monitoring stations in different locations in India for each day of the year 2018 [16]. However, there are many missing entries in this dataset. Given this multidomain data that has spatial and temporal dimensions, the aim is to learn the graph factors that best explain the data. More precisely, we aim to learn the graph factors that capture the relationships between spatially distributed sensors and a graph that captures the seasonal variations of the PM2.5 data. Since the dataset has missing entries, we impute the missing entries using a graph Laplacian regularized nuclear norm minimization [17]. That is, we use

\[
f(X, Y) := \sum_{i=1}^{T} \| A(X_i - Y_i) \|^2 + \gamma \| X_i \|_n, \quad \text{where } A \text{ denotes the known observation mask that selects the available entries, } \| \cdot \|_n \text{ denotes the nuclear norm, and } \gamma \geq 0 \text{ is the tuning parameter.}
\]

As the problem (2) is not convex in \( \{X, L_P, L_Q\} \) due to the coupling between the variables in the smoothness promoting terms, we use alternating minimization method. Specifically, in an alternating manner, we solve for \( \{L_P, L_Q\} \) by fixing \( X \) using the solver described in Section 6, and then solve for \( X \) by fixing \( \{L_P, L_Q\} \) as

\[
\min_X f(X, Y) + \alpha \sum_{i=1}^{T} [\text{tr}(X_i^T L_P X_i) + \text{tr}(X_i L_Q X_i^T)],
\]

where recall that \( \text{vec}(X_i) \) forms the \( i \)th column of \( X \).

The learned graph factors that represent the multidomain graph signals are shown in Fig. 2. Fig. 2a shows the graph factor that encodes the relationships between the spatially distributed air quality monitoring stations. Each colored dot indicates the concentration of PM2.5 on a random day across different locations. We can see from Fig. 2a that the graph connections are not necessarily related to the geographical proximity of the cities, but based on the similarity of the PM2.5 values.
The concentration of PM$_{2.5}$ is relatively lesser during the summer/monsoon months (i.e., June, July, and August) as compared to the other months. The graph capturing these temporal variations in the PM$_{2.5}$ data is shown in Fig. 2b. Each color dot indicates the concentration of PM$_{2.5}$ averaged over each month at a random location. We can see in Fig. 2b that months with similar average concentration of PM$_{2.5}$ are connected. Moreover, the months having low PM$_{2.5}$ values (during the monsoon months) are clustered together and the remaining months are clustered into two groups based on their occurrence before and after the monsoon. This shows that the obtained time graph has clusters that capture the seasonal variations hence confirms the quality of the learned graph.

8. CONCLUSIONS

We developed a framework for learning graphs that can be factorized as the Cartesian product of two smaller graphs. The estimation of the Laplacian matrices of the factor graphs is posed as a convex optimization problem. The proposed solver is computationally efficient and has an explicit solution based on the water-filling method. The performance of the proposed method is significantly better than other intuitive ways to solve the problem. We present numerical experiments based on air pollution data collected across different locations in India. Since this dataset has missing entries, we developed a task-cognizant learning method to solve the joint matrix completion and product graph learning problem.
9. REFERENCES


