Foundations and Trends® in Machine Learning

Data Analytics on Graphs Part III: Machine Learning on Graphs, from Graph Topology to Applications

Suggested Citation: Ljubiša Stanković, Danilo Mandic, Miloš Daković, Miloš Brajović, Bruno Scalzo, Shengxi Li and Anthony G. Constantinides (2020), "Data Analytics on Graphs Part III: Machine Learning on Graphs, from Graph Topology to Applications", Foundations and Trends® in Machine Learning: Vol. 13, No. 4, pp 332–530. DOI: 10.1561/2200000078-3.

Ljubiša Stanković
University of Montenegro
Montenegro
ljubisa@ucg.ac.me

Danilo Mandic
Imperial College London
UK
d.mandic@imperial.ac.uk

Miloš Daković
University of Montenegro
Montenegro
milos@ucg.ac.me

Miloš Brajović
University of Montenegro
Montenegro
milosb@ucg.ac.me

Bruno Scalzo
Imperial College London
UK
bruno.scalzo-dees12@imperial.ac.uk

Shengxi Li
Imperial College London
UK
shengxi.li17@imperial.ac.uk

Anthony G. Constantinides
Imperial College London
UK
a.constantinides@imperial.ac.uk

now
the essence of knowledge
Boston — Delft

Contents

1 Introduction **335**

2 Geometrically Defined Graph Topologies **338**

3 Graph Topology Based on Signal Similarity **347**

4 Learning of Graph Laplacian from Data **355**
4.1 Imposing Sparsity on the Connectivity Matrix 358
4.2 Smoothness Constrained Learning of Graph Laplacian . . . 363
4.3 Graph Topology Estimation with the Graph Laplacian Energy Condition 366
4.4 Learning of Generalized Laplacian-Graphical LASSO 367
4.5 Graph Topology Learning Based on the Eigenvectors . . . 376

5 From Newton Minimization to Graphical LASSO, via LASSO **387**
5.1 Newton Method 387
5.2 Standard LASSO 389
5.3 Graphical LASSO 392

6 Physically Well Defined Graphs **398**
6.1 Resistive Electrical Circuits 398

6.2 Heat Transfer . . . 406
6.3 Spring-Mass Systems . . . 406
6.4 Social Networks and Linked Pages . . . 408
6.5 PageRank . . . 409
6.6 Random Walk . . . 412
6.7 Hitting and Commute Time . . . 416
6.8 Relating Gaussian Random Signal to Electric Circuits . . . 419

7 Graph Learning from Data and External Sources 422

8 Random Signal Simulation on Graphs 428

9 Summary of Graph Learning from Data Using Probabilistic Generative Models 431
9.1 Basic Gaussian Models . . . 432
9.2 Gaussian Graphical Model . . . 434
9.3 Diffusion Models . . . 441

10 Graph Neural Networks 446
10.1 Basic Graph Elements Related to GCNs . . . 448
10.2 Gradient Descent as a Diffusion Process . . . 451
10.3 Label Propagation as a Diffusion Process with External Sources . . . 452
10.4 GNNs of a Recurrent Style . . . 454
10.5 Spatial GCNs via Localization of Graphs . . . 457
10.6 Spectral GCNs via Graph Fourier Transform . . . 459
10.7 Link Prediction via Graph Neural Nets . . . 465

11 Tensor Representation of Lattice-Structured Graphs 473
11.1 Tensorization of Graph Signals in High-Dimensional Spaces 474
11.2 Tensor Decomposition . . . 475
11.3 Connectivity of a Tensor . . . 478
11.4 DFT of a Tensor . . . 479
11.5 Unstructured Graphs . . . 481
11.6 Tensor Representation of Multi-Relational Graphs . . . 482
11.7 Multi-Graph Tensor Networks . . . 484

12 Metro Traffic Modeling Through Graphs **490**
12.1 Traffic Centrality as a Graph-Theoretic Measure 491
12.2 Modeling Commuter Population from Net Passenger Flow 493

13 Portfolio Cuts **499**
13.1 Structure of Market Graph 501
13.2 Minimum Cut Based Vertex Clustering 502
13.3 Spectral Bisection Based Minimum Cut 504
13.4 Repeated Portfolio Cuts 506
13.5 Graph Asset Allocation Schemes 508
13.6 Numerical Example 509

14 Conclusion **512**

Acknowledgments **514**

References **515**

Data Analytics on Graphs Part III: Machine Learning on Graphs, from Graph Topology to Applications

Ljubiša Stanković[1], Danilo Mandic[2], Miloš Daković[3], Miloš Brajović[4], Bruno Scalzo[5], Shengxi Li[6] and Anthony G. Constantinides[7]

[1] *University of Montenegro, Montenegro; ljubisa@ucg.ac.me*
[2] *Imperial College London, UK; d.mandic@imperial.ac.uk*
[3] *University of Montenegro, Montenegro; milos@ucg.ac.me*
[4] *University of Montenegro, Montenegro; milosb@ucg.ac.me*
[5] *Imperial College London, UK; bruno.scalzo-dees12@imperial.ac.uk*
[6] *Imperial College London, UK; shengxi.li17@imperial.ac.uk*
[7] *Imperial College London, UK; a.constantinides@imperial.ac.uk*

ABSTRACT

Modern data analytics applications on graphs often operate on domains where graph topology is not known a priori, and hence its determination becomes part of the problem definition, rather than serving as prior knowledge which aids the problem solution. Part III of this monograph starts by a comprehensive account of ways to learn the pertinent graph topology, ranging from the simplest case where the physics of the problem already suggest a possible graph structure, through to general cases where the graph structure is to be learned from the data observed on a graph. A particular emphasis is placed on the use of standard "relationship measures" in this context, including the correlation and precision matrices, together with the ways to combine these

Ljubiša Stanković, Danilo Mandic, Miloš Daković, Miloš Brajović, Bruno Scalzo, Shengxi Li and Anthony G. Constantinides (2020), "Data Analytics on Graphs Part III: Machine Learning on Graphs, from Graph Topology to Applications", Foundations and Trends® in Machine Learning: Vol. 13, No. 4, pp 332–530. DOI: 10.1561/2200000078-3.

with the available prior knowledge and structural conditions, such as the smoothness of the graph signals or sparsity of graph connections. Next, for learning sparse graphs (that is, graphs with a small number of edges), the utility of the least absolute shrinkage and selection operator, known as (LASSO) is addressed, along with its graph specific variant, the graphical LASSO. For completeness, both variants of LASSO are derived in an intuitive way, starting from basic principles. An in-depth elaboration of the graph topology learning paradigm is provided through examples on physically well defined graphs, such as electric circuits, linear heat transfer, social and computer networks, and spring-mass systems. We also review main trends in graph neural networks (GNN) and graph convolutional networks (GCN) from the perspective of graph signal filtering. Particular insight is given to the role of diffusion processes over graphs, to show that GCNs can be understood from the graph diffusion perspective. Given the largely heuristic nature of the existing GCNs, their treatment through graph diffusion processes may also serve as a basis for new designs of GCNs. Tensor representation of lattice-structured graphs is next considered, and it is shown that tensors (multidimensional data arrays) can be treated as a special class of graph signals, whereby the graph vertices reside on a high-dimensional regular lattice structure. Finally, the concept of graph tensor networks is shown to provide a unifying framework for learning of big data on irregular domains. This part of monograph concludes with an in-dept account of emerging applications in financial data processing and underground transportation network modeling. More specifically, by means of portfolio cuts of an asset graph, we show how domain knowledge can be meaningfully incorporated into investment analysis, while the underground transportation example addresses

vulnerability of stations in the London underground network to traffic disruption.

Keywords: graph theory; random data on graphs; big data on graphs; signal processing on graphs; machine learning on graphs; graph topology learning; systems on graphs; vertex-frequency estimation; graph neural networks; graphs and tensors.

1

Introduction

Graph data analytics have already shown enormous potential, as their flexibility in the choice of graph topologies (irregular data domains) and connections between the entities (vertices) allows for both a rigorous account of irregularly spaced information such as locations and social connections, and also for the incorporation of semantic and contextual cues, even for otherwise regular structures such as images.

In Part I and Part II of this monograph, it was assumed that the graph itself is already defined prior to analyzing data on graphs. The focus of Part I has been on defining graph properties through the mathematical formalism of linear algebra, while Part II introduces graph counterparts of several important standard data analytics algorithms, again for a given graph. However, in many modern applications, graph topology is not known a priori (Cioacă *et al.*, 2019; Das *et al.*, 2017; Dong *et al.*, 2015, 2016; Epskamp and Fried, 2018; Friedman *et al.*, 2008; Hamon *et al.*, 2019, Meinshausen *et al.*, 2006; Pavez and Ortega, 2016; Pourahmadi, 2011; Rabiei *et al.*, 2019; Stanković *et al.*, 2018, 2020), and the focus of this part is therefore on simultaneous estimation of data on a graph and the underlying graph topology. Without loss of generality, it is convenient to assume that the vertices are given, while

the edges and their associated weights are part of the solution to the problem considered and need to be estimated from the vertex geometry and/or the observed data (Bohannon *et al.*, 2019; Caetano *et al.*, 2009; Camponogara and Nazari, 2015; Dal Col *et al.*, 2019; Gu and Wang, 2019; Mao and Gu, 2019; Pasdeloup *et al.*, 2019; Slawski and Hein, 2015; Segarra *et al.*, 2016; Stanković and Sejdić, 2019; Stanković *et al.*, 2017; Tanaka and Sakiyama, 2019; Thanou *et al.*, 2014; Ubaru *et al.*, 2017; Yankelevsky and Elad, 2016; Zhao *et al.*, 2012; Zheng *et al.*, 2011).

Three scenarios for the estimation of graph edges from vertex geometry or data are considered in this part of the monograph.

- Based on the *geometry of vertex positions*. In various sensor network setups (such as temperature, pressure, and transportation), the locations of the sensing positions (vertices) are known beforehand, while the vertex distances convey physical meaning about data dependence and thus may be employed for edge/weight determination.

- Based on *data association and data similarity.* Various statistical measures are available to serve as data association metrics, with the covariance and precision matrices most commonly used. A strong correlation between data on two vertices would indicate a large weight associated with the corresponding edge. A small degree of correlation would indicate nonexistence of an edge (after weight thresholding).

- *Based on physically well defined relations among the sensing positions*. Examples include electric circuits, power networks, linear heat transfer, social and computer networks, spring-mass systems, to mention but a few. In these cases, edge weighting can usually be well defined based on the underlying context of the considered problem.

After a detailed elaboration of graph definition and graph topology learning techniques, a summary of graph topology learning from data using probabilistic generative models is given. This followed by an account of graph neural networks (GNN), with a special emphasis on

graph convolutional networks (GCN). The analysis is considered from the perspective of graph signal filtering presented in Part II. Graph data analysis is further generalized to the tensor representation of lattice-structured graphs, whereby the graph vertices reside on a high-dimensional tensor structure. Finally, two applications of graph-based data analysis are given: (i) an example where domain knowledge is incorporated into financial data analysis (the investment analysis), by means of portfolio cuts; (ii) London underground transportation system. The latter example demonstrates how graph theory can be used to identify the stations in the London underground network which have the greatest influence on the functionality of the traffic, and also to assess the impact of a station closure on service levels across the city.

2

Geometrically Defined Graph Topologies

For a graph that corresponds to a network with geometrically distributed vertices, it is natural to relate the edge weights with the distance between vertices. Consider vertices m and n whose locations in space are defined by the position vectors (coordinates) $\mathbf{r}_m$ and $\mathbf{r}_n$. The Euclidean distance, r_{mn}, between these two vertices is then

$$r_{mn} = \text{distance}(m, n) = \|\mathbf{r}_m - \mathbf{r}_n\|_2.$$

A common way to define the graph weights in such networks is through an exponentially decaying function of the distance, r_{mn}, for example as

$$W_{mn} = \begin{cases} e^{-r_{mn}^2/\tau^2}, & \text{for } r_{mn} \leq \kappa \\ 0, & \text{for } r_{mn} > \kappa \text{ or } m = n, \end{cases} \tag{2.1}$$

where τ and κ are suitably chosen constants. This is also physically well justified, as based on $e^{-r_{mn}^2/\tau^2}$ the weights tend to 1 for closely spaced vertices and diminish for distant vertices.

The rationale for this definition of edge weights is the assumption that the signal value measured at a vertex n is similar to signal values measured at its neighboring vertices. Then, the estimation of a signal at a vertex n should also involve neighboring vertices which are connected

with larger weights (close to 1), while the signal values sensed at farther vertices would be less relevant, and are associated with smaller weighting coefficients or are not included at all. A physical interpretation of the weights in (2.1), within the heat distribution and the heat kernel frameworks, can be found in Belkin and Niyogi (2003), where the constant $\tau^2 = 4t$ is considered as the heat kernel parameter, t. Moreover, the Laplacian induced from such weight definition can converge to the continuous Laplace–Beltrami operator if the data is random and uniformly distributed and the number of data point is infinite (Belkin and Niyogi, 2008).

The Gaussian function, used in (2.1), is appropriate in many applications, however, other forms to penalize data values associated with the vertices which are far from the considered vertex may also be used. Examples of such functions include various kernels, such as the kernel given by Chen *et al.* (2015, 2016)

$$W_{mn} = \begin{cases} e^{-r_{mn}/\tau}, & \text{for } r_{mn} \leq \kappa \\ 0, & \text{for } r_{mn} > \kappa \text{ or } m = n \end{cases} \tag{2.2}$$

or the inverse Euclidean distance between vertices m and n, given by

$$W_{mn} = \begin{cases} \dfrac{1}{r_{mn}}, & \text{for } r_{mn} \leq \kappa \\ 0, & \text{for } r_{mn} > \kappa \text{ or } m = n. \end{cases} \tag{2.3}$$

Obviously, the simplest form for the edge weighting coefficients is a binary scheme

$$W_{mn} = A_{mn} = \begin{cases} 1, & \text{for } r_{mn} \leq \kappa \\ 0, & \text{for } r_{mn} > \kappa \text{ or } m = n, \end{cases} \tag{2.4}$$

which corresponds to an unweighted graph, with $\mathbf{W} = \mathbf{A}$. This form can be obtained from (2.1) as $\tau^2 \to \infty$ (or the heat kernel parameter approaches infinity).

Example 1: We shall illustrate the geometry-based formation of graph structure on the well-known Swiss roll manifold as a domain for data acquisition. This is a three-dimensional surface with the space coordinates, (x, y, z), defined as functions of two parameters, ξ and ζ, in the

following form

$$\begin{aligned} x &= \frac{1}{4\pi}\zeta\cos(\zeta) \\ y &= \xi \\ z &= \frac{1}{4\pi}\zeta\sin(\zeta). \end{aligned} \tag{2.5}$$

The Swiss roll manifold shown in Figure 2.1(a) was created for the parameters, ξ and ζ, ranging within the intervals $-1 \leq \xi \leq 1$ and $\pi \leq \zeta \leq 4\pi$.

More specifically, we considered a graph with $N = 100$ vertices, which were randomly placed on the Swiss roll surface, with the coordinates (x_k, y_k, z_k), $k = 1, 2, \ldots, N$, whereby

$$\begin{aligned} \xi_k &\text{ was uniformly random within } -1 \leq \xi_k \leq 1 \\ \zeta_k &\text{ was uniformly random within } \pi \leq \zeta_k \leq 4\pi. \end{aligned}$$

The vertices were connected with edges, with the corresponding edges defined as in (2.1), that is

$$W_{mn} = \exp(-r_{mn}^2/\tau^2),$$

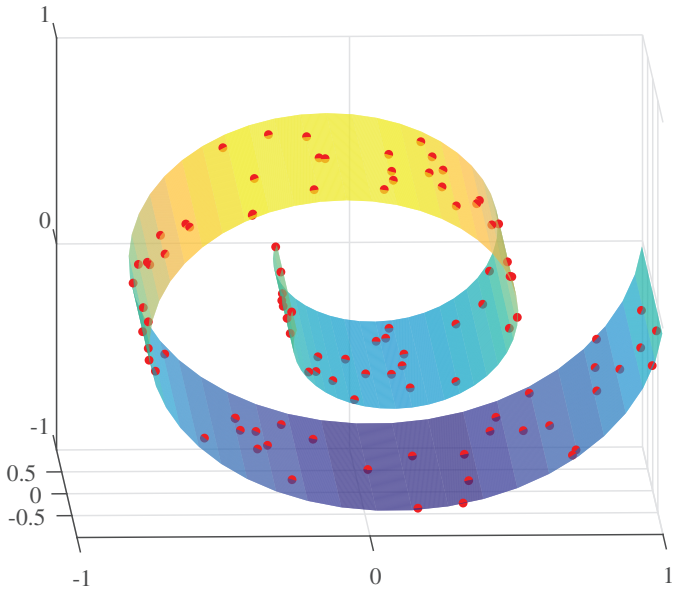
for $r_{mn} > 0.6$, with $W_{mn} = 0$ for $r_{mn} \leq 0.6$, as well as for $m = n$; $\tau = 1/2$. The symbol r_{mn} denotes the shortest geodesic distance between the vertices m and n, measured along the Swiss roll manifold, in the following way

$$r_{mn}^2 = l_{mn}^2 + (y_m - y_n)^2,$$

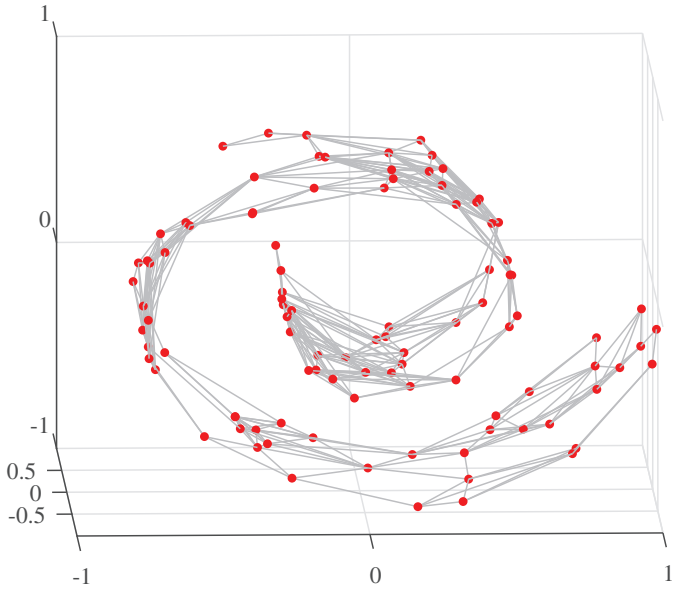
where the arc length, l_{mn}, of the parametric curve in (2.5) is

$$\begin{aligned} l_{mn} &= \int_{\zeta_m}^{\zeta_n} \sqrt{\left(\frac{dx}{d\zeta}\right)^2 + \left(\frac{dz}{d\zeta}\right)^2}\, d\zeta \\ &= \frac{1}{4\pi}\int_{\zeta_m}^{\zeta_n} \sqrt{\left(\frac{d(\zeta\cos(\zeta))}{d\zeta}\right)^2 + \left(\frac{d(\zeta\sin(\zeta))}{d\zeta}\right)^2}\, d\zeta \\ &= \frac{1}{4\pi}\int_{\zeta_m}^{\zeta_n} \sqrt{1+\zeta^2}\, d\zeta = \frac{1}{4\pi}\left(\frac{1}{2}\zeta\sqrt{\zeta^2+1} + \frac{1}{2}\ln\left(\sqrt{\zeta^2+1}+\zeta\right)\right)\Bigg|_{\zeta_m}^{\zeta_n}. \end{aligned}$$

Small weight values were hard-thresholded to zero, in order to reduce the number of edges associated with each vertex.

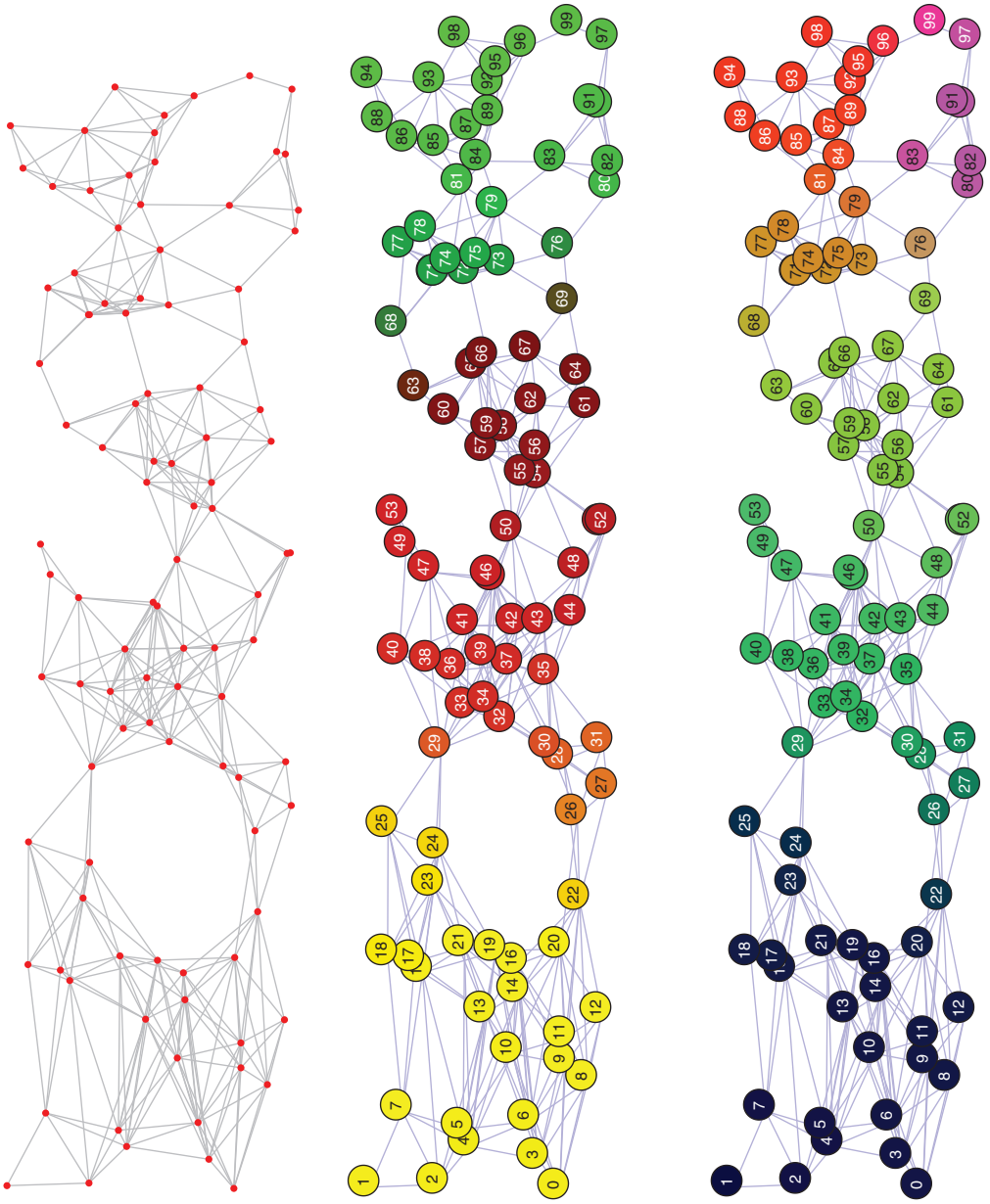


(a)



(b)

Figure 2.1: Concept of graph definition based on problem geometry. (a) Vertices (points) on a three-dimensional manifold called the Swiss roll surface. (b) A graph representation on the Swiss roll manifold. (c) Two-dimensional presentation of the three-dimensional graph from (b) obtained by unfolding the original 3D surface. (d) Vertices colored using the spectral vector, $\mathbf{q}_n = [u_1(n), u_2(n)]$, formed from the two smoothest *generalized eigenvectors* of the graph Laplacian, $\mathbf{u}_1$ and $\mathbf{u}_2$. (e) Vertices colored using the spectral vector, $\mathbf{q}_n = [u_1(n), u_2(n), u_3(n)]$, formed from the three smoothest *eigenvectors* of the graph Laplacian, $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$. The vertex indexing in (d) and (e) is performed based on the sorted values of the smoothest (Fiedler) eigenvector, $\mathbf{u}_1$.



(c) (d) (e)

Figure 2.1: Continued.

The so produced three-dimensional graph is shown in Figure 2.1(b), and its two-dimensional presentation in Figure 2.1(c). The vertices were ordered so that the values of the Fiedler eigenvector, $u_1(n)$, were nondecreasing; the vertices were colored based on the two-dimensional and three-dimensional spectral vectors, $\mathbf{q}_n = [u_1(n), u_2(n)]$ and $\mathbf{q}_n = [u_1(n), u_2(n), u_3(n)]$, of the Swiss roll in Figures 2.1(d) and (e). This

kind of vertex indexing can also be used for clustering with, for example, the k-means clustering presented in Part I, Remark 30.

Classical Gaussian filter within graph topology formulation. To illustrate this classical operation on the discrete-time domain data, assume that we desire to perform classical smoothing of a discrete-time domain signal, $x(n)$, at a vertex/instant n, through a moving average operation on data observed at neighboring vertices/instants, $x(m)$, using a truncated Gaussian weighting function given by

$$g(m,n) = e^{-(m-n)^2/\tau^2}$$

for $|m-n| \leq \kappa$ and $g(m,n) = 0$ for $|m-n| > \kappa$. The smoothed discrete-time domain signal, $y(n)$, can be expressed in classical data analysis as

$$y(n) = \sum_m e^{-\frac{(m-n)^2}{\tau^2}} x(m) \tag{2.6}$$

where the summation is performed for instants/vertices, m, such that $|n-m| \leq \kappa$.

We shall now reformulate this classical data processing problem within the graph topology framework. The distance between the sampling instants/vertices, distance (m,n), plays a crucial role in signal smoothing, and is defined as

$$\text{distance}(m,n) = r_{mn} = \|m-n\|_2 = |m-n|.$$

The corresponding edge weights can be defined based on the Gaussian smoothing function, and are given by $W_{mn} = e^{-r_{mn}^2/\tau^2}$ for $r_{mn} \leq \kappa$, and $W_{mn} = 0$ for $r_{mn} > \kappa$ and $m = n$.

The classical smoothed signal, $y(n)$, defined in (2.6) can now be expressed in the form appropriate for the graph framework as

$$y(n) = x(n) + \sum_m x(m) W_{mn} = x(n) + \sum_m e^{-\frac{(m-n)^2}{\tau^2}} x(m)$$

where the summation is performed for vertices m such that $|m-n| \leq \kappa$ and $m \neq n$. This operation can be defined within the graph analysis framework as a simple first order system on graph, given by

$$\mathbf{y} = \mathbf{W}^0\mathbf{x} + \mathbf{W}^1\mathbf{x}$$

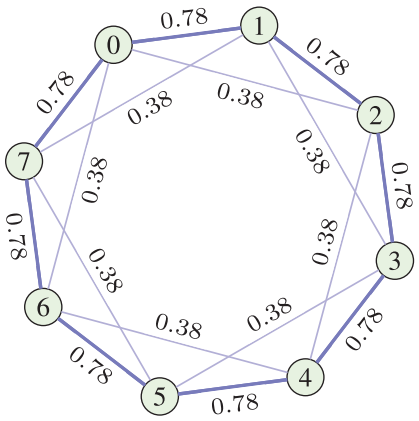


Figure 2.2: Graph which corresponds to the weighted moving average operator with Gaussian weights given in (2.6).

where the edge weights between the vertices m and n are defined by W_{mn}.

For example, for $\tau = 2$ and $\kappa = 2$, the edge weights W_{mn} are shown in Figure 2.2 and this graph-based formulation is identical to the classical discrete-time domain weighted moving average

$$y(n) = x(n) + \sum_{m} W_{mn} x(m) = \sum_{m=n-2}^{n+2} e^{-\frac{(m-n)^2}{4}} x(m), \tag{2.7}$$

with the output signal samples, $y(n)$, equal to the output of a first-order system on the graph given by

$$\mathbf{y} = \mathbf{W}^0\mathbf{x} + \mathbf{W}^1\mathbf{x} = 3.29\mathbf{L}^0\mathbf{x} - \mathbf{L}^1\mathbf{x},$$

where $\mathbf{W}^0$ and $\mathbf{L}^0$ are identity matrices, by definition.

For image input data, where the vertices correspond to the pixel positions and the Euclidean distance between pixels is used to model the image domain as a graph, the previous example would model a moving average filtered image, using a radial Gaussian window.

Example 2: Consider the benchmark Minnesota roadmap graph, for which the connectivity map (adjacency matrix) is designated by the road connections and the vertices are located at the road crossings. The edges are defined by the adjacency matrix and were weighted according to their Eucledian distances using the weighting scheme in (2.2), with

$\tau = 25$ km, to give

$$W_{mn} = e^{-r_{mn}^2/\tau^2},$$

where the threshold κ was not used since the connectivity is already determined by the given adjacency matrix.

We considered a simulated temperature signal in the Minnesota area (normalized temperature filed) which was calculated as

$$x(n) = 0.9\Bigg(0.1 + 0.8e^{-\left(\frac{x-150}{100}\right)^2-\left(\frac{y-400}{200}\right)^2}, \\ + 0.5e^{-\left(\frac{x-450}{200}\right)^2-\left(\frac{y-400}{100}\right)^2} + e^{-\left(\frac{x-500}{250}\right)^2-\left(\frac{y-150}{200}\right)^2}\Bigg) + \nu(n)$$

where $\nu(n)$ is white Gaussian noise with a standard deviation $\sigma_\nu = 0.3$. The noise-free and noisy versions of this graph temperature signal are given respectively in Figures 2.3(a) and (b). The noisy signal was filtered in the vertex domain by a low-pass filter implemented using Taubin's $\alpha - \beta$ algorithm (presented in Part II, Section 6.2) with $\alpha = 0.15$ and $\beta = 0.1$, and the so enhanced temperature signal is shown in Figure 2.3(c). The output SNR of 19.34 dB was achieved for the input SNR of 9.35 dB, a gain of 10 dB.

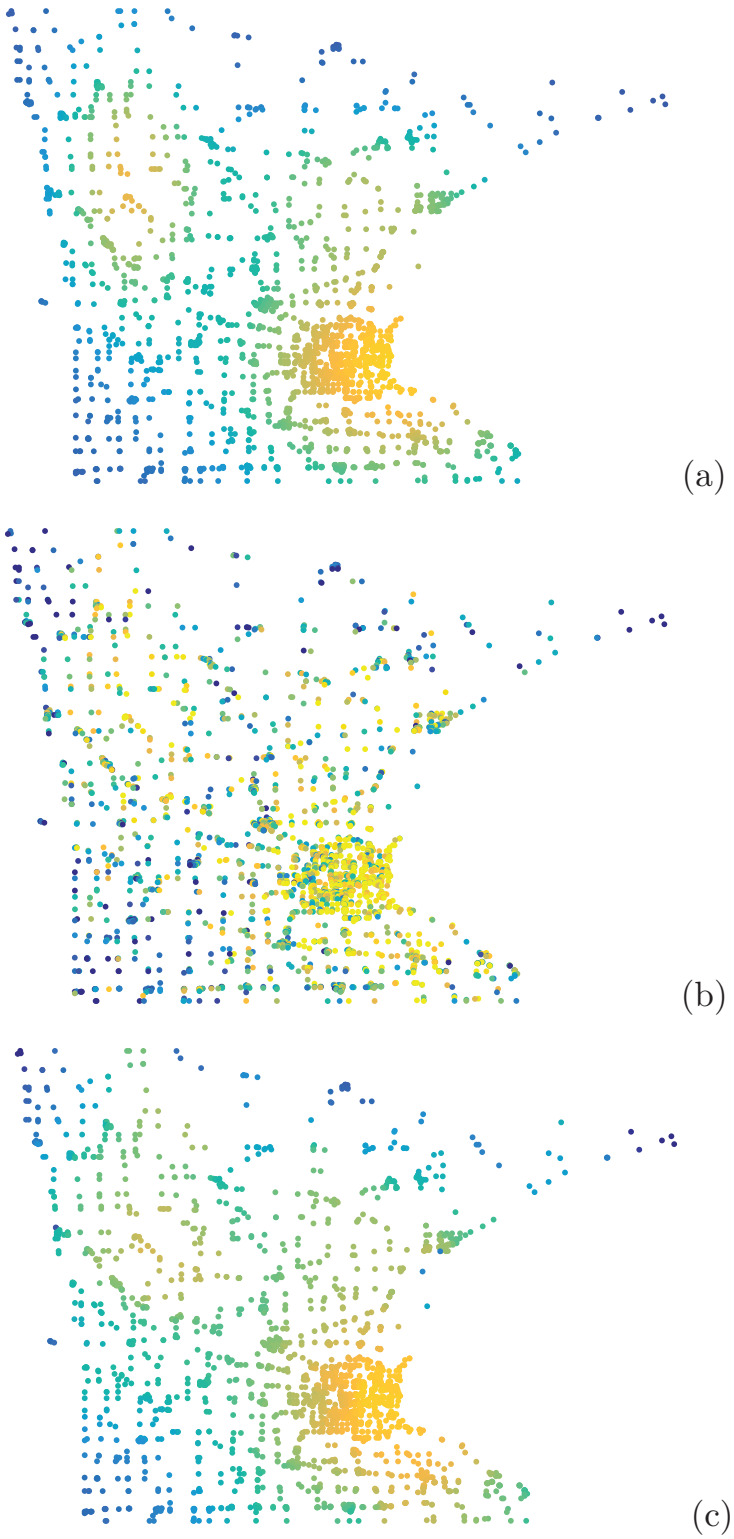


Figure 2.3: Temperatures simulated on the Minnesota roadmap graph. (a) Original synthetic temperature field signal. (b) Noisy temperature signal. (c) Low-pass filtered temperature signal from (b). The signal values are designated by the corresponding vertex color.

3

Graph Topology Based on Signal Similarity

In the previous sections, graph weights were defined on the assumption that the geometric distance of vertices, where the signal is sensed, is a reliable indicator of data similarity, or some other more general data association. Indeed, this is the case with, for example, the measurements of atmospheric temperature and (barometric) pressure when the terrain configuration has no influence on the similarity of measured data. However, in general, the geometric distance between vertices may not be a good indicator of data similarity.

One such example is in image processing, where the pixel color values themselves can be used as an indicator of signal similarity; this can be achieved in combination with the distances between pixels, which play the role of vertices. If the intensity values at pixels indexed by m and n are denoted by $x(m)$ and $x(n)$, then the difference of intensities is defined by

$$\text{Intensity}_{\text{distance}}(m,n) = r_{mn} = |x(m) - x(n)|,$$

and the corresponding weights may be defined as

$$W_{mn} = \begin{cases} e^{-(x(m)-x(n))^2/\tau^2}, & \text{for } r_{mn} \leq \kappa \text{ and } \rho_{mn} \leq \gamma \\ 0, & \text{for } r_{mn} > \kappa \text{ or } \rho_{mn} > \gamma \text{ or } m = n, \end{cases}$$

where ρ_{mn} is a geometric distance between the considered pixels/vertices and τ, κ, and γ are chosen constants.

More reliable measures of data similarity can be defined when it is possible to collect more than one snapshot of data for a given set of sensing points/vertices. Assume that at every vertex $n = 0, 1, \ldots, N-1$ we have acquired P signal values, denoted by $x_p(n),\ p = 1, 2, \ldots, P$. Such a dataset may be equally treated as multivariate data or signal measurements in a sequence. Then, an appropriate similarity measure for a real-valued signal at vertices m and n may be

$$r_{mn}^2 = \frac{\sum_{p=1}^{P}(x_p(m) - x_p(n))^2}{\sum_{m=1}^{N-1}\sum_{n=1}^{N-1}\sum_{p=1}^{P}(x_p(m) - x_p(n))^2} \tag{3.1}$$

so that $\sum_{m=1}^{N-1}\sum_{n=1}^{N-1} r_{mn}^2 = 1$.

The graph weights can again be defined using any of the previous forms, for example, as

$$W_{mn} = \begin{cases} e^{-r_{mn}^2/\tau^2}, & \text{for } r_{mn} \le \kappa \\ 0, & \text{for } r_{mn} > \kappa \text{ or } m = n, \end{cases}$$

or

$$W_{mn} = \begin{cases} e^{-r_{mn}/\tau}, & \text{for } r_{mn} \le \kappa \\ 0, & \text{for } r_{mn} > \kappa \text{ or } m = n. \end{cases}$$

The geometric distance between the considered pixels/vertices, ρ_{mn}, can also be included in the weight definition.

Random observations. When the signal values, $x_p(n)$, acquired over P observations, $p = 1, 2, \ldots, P$ at N vertices $n = 0, 1, \ldots, N-1$, are drawn from zero-mean random noise with equal variances, $\sigma_x^2 = 1$, the similarity measure can be defined by

$$r_{mn}^2 = \frac{\sum_{p=1}^{P}(x_p(m) - x_p(n))^2}{\sqrt{\sum_{p=1}^{P} x_p^2(m) \sum_{p=1}^{P} x_p^2(n)}} = 2(1 - R_x(m, n))$$

where

$$R_x(m, n) = \frac{1}{P}\sum_{p=1}^{P} x_p(m)x_p(n)$$

represents the normalized sample autocorrelation function and $\sigma_x^2 = \frac{1}{P}\sum_{p=1}^{P} x_p^2(n) = 1$ for sufficiently large P.

Similarity metrics for images. The same structure can be used for other applications, such as in image classification or handwritten letter recognition. In these cases, the *distance between an image m and an image n* is equal to

$$r_{mn} = \text{Image}_{\text{distance}}(m, n) = \|\mathbf{x}_m - \mathbf{x}_n\|_F, \tag{3.2}$$

where

$$\|\mathbf{x}\|_F = \sqrt{\sum_m \sum_n |x(m, n)|^2},$$

is the Frobenius norm of an image matrix $\mathbf{x}$ (that is, the square root of the sum of squared image values over all pixels).

Block collaborative image processing. A class of recent efficient image processing algorithms is based on detecting similar blocks within an image, followed by collaborative processing using those similar blocks. Image enhancement algorithms then assume that the basic images are also similar within these blocks, while the corresponding noise is not related and can be averaged out. The similarity between the image blocks, $\mathbf{x}_m$ and $\mathbf{x}_n$, may then be defined similar to (3.2), using their distance given by

$$r_{mn} = \text{Block}_{\text{distance}}(m, n) = \|\mathbf{x}_m - \mathbf{x}_n\|_F.$$

The similarity among the blocks in an image can be modeled by a graph, and such graph models may be used as bases for collaborative processing of image blocks. Recall that a block of $B \times B$ pixels is an example of a vertex in a B^2-dimensional space, since it is defined by $B \times B$ independent pixel values (vertex coordinates/dimensions).

Generalized distance measure. The Euclidean distance is typically used in the calculation of the distance between two blocks of data, $\mathbf{x}_m$ and $\mathbf{x}_n$. It may be generalized by introducing the inner product matrix, $\mathbf{H}$, into distance calculation to yield

$$r_{mn}^2 = (\mathbf{x}_m - \mathbf{x}_n)^T \mathbf{H} (\mathbf{x}_m - \mathbf{x}_n),$$

where the data sets $\mathbf{x}_m$ and $\mathbf{x}_n$ are represented in the column vector form. When the inner product matrix, $\mathbf{H}$, is an identity matrix, $\mathbf{H} = \mathbf{I}$, the standard Euclidean distance is obtained. If we use, for example, $\mathbf{H} = \mathbf{U}_C\mathbf{U}_C^T$, where $\mathbf{U}_C$ is the matrix with cosine transform basis functions as its columns, we will arrive at

$$\begin{aligned} r_{mn}^2 &= (\mathbf{x}_m - \mathbf{x}_n)^T \mathbf{U}_C \mathbf{U}_C^T (\mathbf{x}_m - \mathbf{x}_n) \\ &= (\mathbf{C}_m - \mathbf{C}_n)^T (\mathbf{C}_m - \mathbf{C}_n) = \|\mathbf{C}_m - \mathbf{C}_n\|_2^2, \end{aligned}$$

where $\mathbf{C}_n$ is the 2D discrete cosine transform (2D DCT) of $\mathbf{x}_n$, written in a vector column format. By virtue of this representation, problem dimensionality can straightforwardly be reduced using only the K slowest-varying basis functions, $\mathbf{U}_C^{(K)}$, instead of the full 2D DCT transformation matrix (this operation corresponds to low-pass filtering of $\mathbf{x}_n$ in the 2D DCT domain, by keeping the K slowest-varying coefficients). In this case, the distance, r_{mn}^2, is of the form

$$\begin{aligned} r_{mn}^2 &= (\mathbf{x}_m - \mathbf{x}_n)^T \mathbf{U}_C^{(K)} \mathbf{U}_C^{(K)^T} (\mathbf{x}_m - \mathbf{x}_n) \\ &= \|\mathbf{C}_m^{(K)} - \mathbf{C}_n^{(K)}\|_2^2, \end{aligned}$$

and is calculated based on the reduced original dimensionality of $\mathbf{x}_n$ or $\mathbf{C}_n$ to the dimensionality K of $\mathbf{C}_n^{(K)}$.

Another interesting form of the inner product matrix is the inverse covariance matrix $\mathbf{H} = \mathbf{\Sigma}^{-1}$, which will be discussed later in Sections 4.4 and 6.8.

Example 3: A noisy image with a designated set of 29 blocks of pixels is shown in Figure 3.1(a). The similarity between any two of the blocks was defined based on the distance

$$r_{mn}^2 = \frac{1}{B^2}\|\mathbf{C}_m - \mathbf{C}_n\|_F^2,$$

where $\mathbf{C}_n$ represents the matrix form of the 2D DCT of the image block $\mathbf{x}_n$.

The 2D DCT was then hard-thresholded, with a threshold equal to $0.1 \max |\mathbf{C}_n|$, to reduce the influence of noise (and dimensionality problems), that is, all 2D DCT coefficients bellow this threshold were

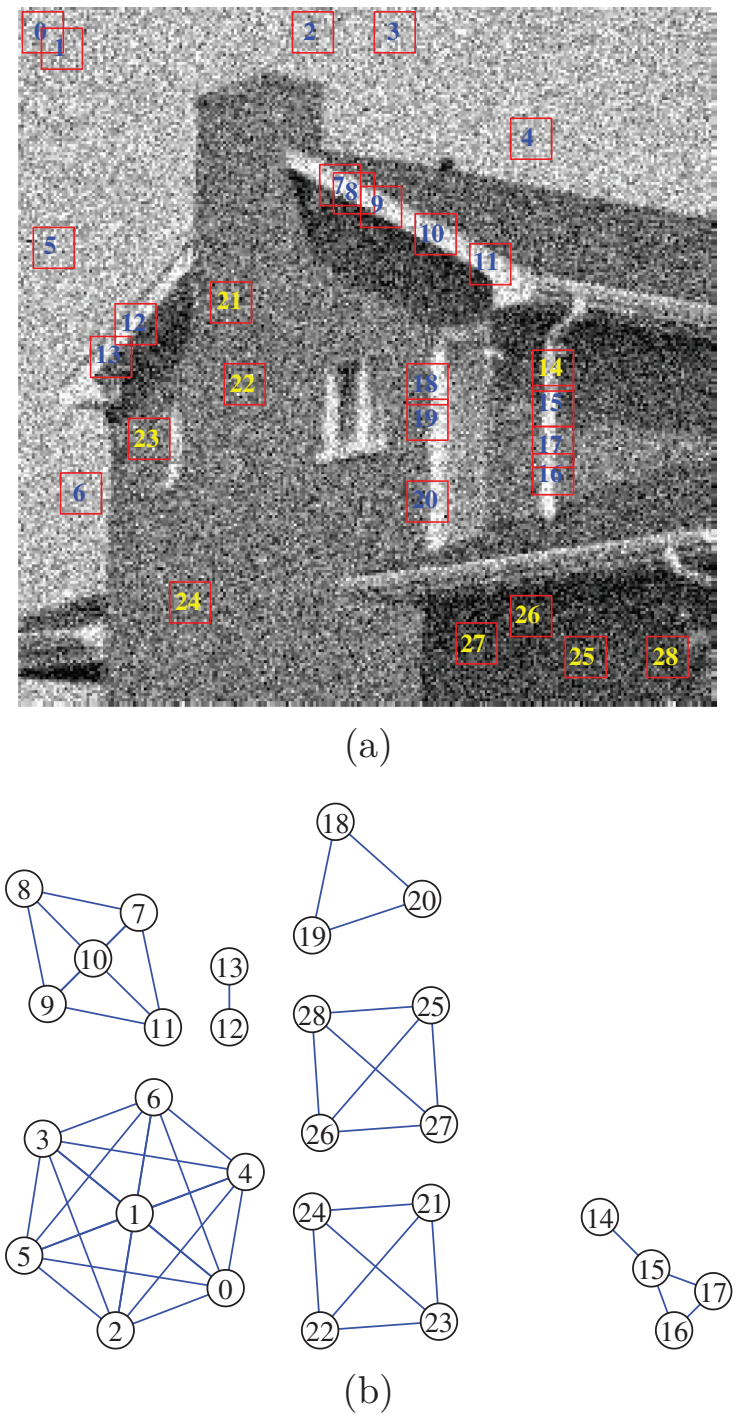


Figure 3.1: Graph learning based on the similarity of blocks of image data. (a) Original image with designated blocks of pixels. (b) The graph produced from the blocks in (a). Notice that the resulting graph consists of seven disconnected subgraphs, which correspond to the seven different groups of blocks.

set to zero, to give

$$C_n(k,l) = \begin{cases} C_n(k,l), & \text{if } |C_n(k,l)| > 0.1 \max |\mathbf{C}_n| \\ 0, & \text{elsewhere.} \end{cases}$$

The edge weights, W_{mn}, for a graph representation of the considered blocks (as vertices) were then calculated as

$$W_{mn} = \exp(-r_{mn}^2 B),$$

for $r_{mn} \leq 0.26$, and $W_{mn} = 0$ for $r_{mn} > 0.26$, or $m = n$, with $B = 16$.

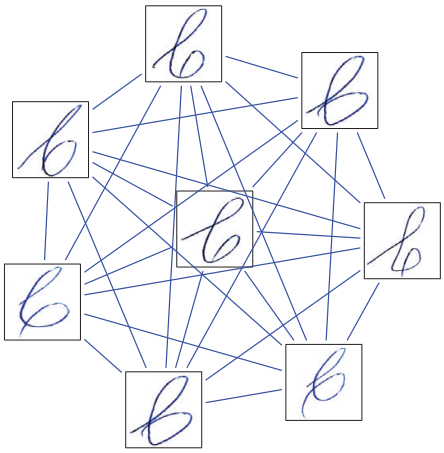
The so obtained graph, which indicates block similarity, is given in Figure 3.1(b). This graph representation is very convenient for collaborative image processing, since the graph structure will ensure that the processing is performed independently on the sets of blocks which share relevant information (connected subgraphs). Notice that the blocks within each subgraph can be considered as a 3D signal of RGB components. Then, for example, a simple averaging over similar blocks (within one subgraph), will not significantly degrade the image detail, while at the same time it will reduce the corresponding noise, as it is uncorrelated in different blocks.

This is precisely the principle of the Block-Matching and 3D filtering (BM3D) algorithm, where the noise and the image are estimated from the set of similar blocks (in our example, from the blocks within a subgraph). The estimation of the related set of blocks in the image and the estimation of noise power is then used to define the Wiener filter. Such Wiener filter is used to filter all related blocks (within the subgraph). The procedure is repeated for each set of similar blocks (subgraphs). Of course, in the case of the BM3D algorithm, for each considered (reference) block, $\mathbf{x}_n$, it is desirable to search over the whole image and to find as many similar blocks as possible in order to obtain the best possible Wiener filter and consequently achieve maximum possible noise reduction.

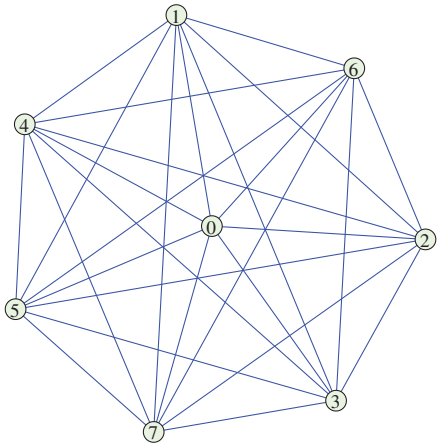
In this example, the blocks and the threshold for edge weights, W_{mn}, were selected so as to produce disconnected graph components and a clear segmentation scheme. If this was not the case, vertex clustering and graph segmentation could be performed using the theory presented in Part I.

Recall that in Part I, Example 24 the *structural similarity index (SSIM)*, was used instead of the simple difference/distance, to relate and cluster images.

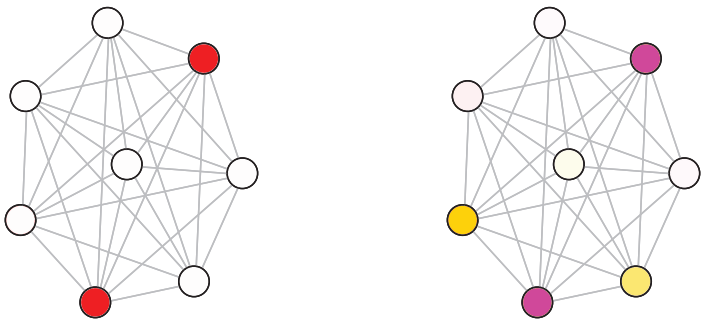
Example 4: Eight images with the hand-written letter "b" were considered and the task was to create their graph representation. The SSIM was calculated for each pair of images and the edge-weights were equal to the calculated SSIM values, as shown in Figure 3.2(a). For the graph from Figure 3.2(b), the generalized eigenvectors of the Laplacian were calculated and the vertices were colored using first the smoothest (Fiedler) eigenvector, $\mathbf{u}_1$, and then using the two smoothest eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$, as a basis for image clusterings, as respectively shown in Figure 3.2(c) (left) and (right).



(a)



(b)



(c)

Figure 3.2: Graph representation of a set of hand-written images of the letter "b". The images serve as vertices, while the weight matrix for the edges is defined through the structural similarity index metric (SSIM) between the images, with $W_{mn} = \text{SSIM}(m, n)$. The vertices are colored in (c) using first the smoothest (Fiedler) eigenvector, $\mathbf{u}_1$, and then the two smoothest eigenvectors, $\mathbf{u}_1$ and $\mathbf{u}_2$, of the generalized eigenvectors of the Laplacian (with the corresponding spectral vectors $\mathbf{q}_n = [u_1(n)]$ and $\mathbf{q}_n = [u_1(n), u_2(n)]$) respectively shown in Figure 3.2(c) (left) and (right).

4

Learning of Graph Laplacian from Data

Consider a graph signal for which we have available P independent observations. Denote by $x_p(n)$ the observed signal at a vertex, n, and for an observation, p. The column vector with graph signal samples from the pth observation is denoted by $\mathbf{x}_p$. All observations from this graph signal can then be arranged into an $N \times P$ matrix, given by

$$\mathbf{X}_P = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P].$$

Designate the $(n+1)$-th row of this matrix by a row vector, $\mathbf{y}_n$, which corresponds to the vertex n, that is

$$\mathbf{y}_n = [x_1(n), x_2(n), \ldots, x_P(n)]. \tag{4.1}$$

Then, the matrix of observations can also be written as

$$\mathbf{X}_P = \begin{bmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{N-1} \end{bmatrix}.$$

The correlation coefficient between vertices m and n, estimated by averaging over the set of P observations, is then given by

$$R_x(m,n) = \frac{1}{P}\sum_{p=1}^{P} x_p(m)x_p(n) = \frac{1}{P}\mathbf{y}_m\mathbf{y}_n^T$$

or in a matrix form

$$\mathbf{R}_x = \frac{1}{P}\mathbf{X}_P\mathbf{X}_P^T. \tag{4.2}$$

If the observations are not zero-mean, then we should use the covariance matrix, $\mathbf{\Sigma}$, with elements

$$\Sigma_x(m,n) = \frac{1}{P}\sum_{p=1}^{P}(x_p(m)-\mu(m))(x_p(n)-\mu(n)), \tag{4.3}$$

where $\mu(n)$ is the mean of the observations at the vertex n.

Remark 1: Since the correlation matrix in (4.2) includes contribution from signals at all vertices, it accumulates correlations obtained through all possible walks from the current vertex, n, to any other vertex, m. This also means that the correlation coefficient between two vertices will produce misleading results if there exists one or more other vertices, q, where the signal is strongly correlated with both of the considered vertices, m and n. This is why the naive use of correlation tends to overestimate the strength of direct vertex connections; this renders it a poor metric for establishing direct links (edges) between vertices. To resolve this issue, either additional conditions should be imposed on the correlation matrix, or other statistical parameters may be used for edge weight estimation.

Example 5: Consider four random graph signals observed at the vertices $n = 0, 1, 2, 3$, and given by

$$\begin{aligned} x_p(0) &= \nu_0(p) \\ x_p(1) &= x_p(0) + \nu_1(p) \\ x_p(2) &= x_p(1) + \nu_2(p) \\ x_p(3) &= x_p(2) + \nu_3(p), \end{aligned} \tag{4.4}$$

where $\nu_0(p), \nu_1(p), \nu_2(p), \nu_3(p)$ are mutually uncorrelated, white random variables with zero mean and unit variance. The elements of the

correlation matrix for the above signals can be calculated as, for example

$$R_x(0,1) = \mathrm{E}\{x_p(0)x_p(1)\} = \mathrm{E}\{x_p(0)(x_p(0) + \nu_1(p))\} = 1$$

or

$$\begin{aligned} R_x(0,2) &= \mathrm{E}\{x_p(0)x_p(2)\} = \mathrm{E}\{x_p(0)(x_p(1) + \nu_2(p))\} \\ &= \mathrm{E}\{x_p(0)(x_p(0) + \nu_2(p) + \nu_2(p))\} = 1. \end{aligned}$$

Observe from (4.4) that, although the signal value $x_p(2)$ is not directly related to $x_p(0)$, the correlation coefficient, $R_x(0,2)$, is nonzero and even equal to $R_x(0,1)$, since there is an indirect link between these two signal values through $x_p(1)$. In practical applications, it is therefore desirable to avoid this indirect cumulative contribution to the correlation coefficient which results in an overestimated edge weight.

All correlation coefficients for the above example can be written in a matrix form as

$$\mathbf{R}_x = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix}, \tag{4.5}$$

with the inverse correlation matrix, called the *precision matrix*, as

$$\mathbf{C} = \mathbf{R}_x^{-1} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}. \tag{4.6}$$

Remark 2: Observe that while the autocorrelation in (4.5) overestimates the strength of edge links, the precision matrix in (4.6) produces the desired results, since for example, $C(0,2) = 0$, which indicates that there is no direct relation between $x_p(0)$ and $x_p(2)$, although $x_p(2)$ is indirectly linked to $x_p(0)$ through $x_p(1)$.

Similar to the normalized correlation, the normalized precision matrix, $\mathbf{C}^{(N)}$, is defined by $C_{mn}^{(N)} = C_{mn}/\sqrt{C_{mm}C_{nn}}$ to produce

$$\mathbf{C}^{(N)} = \begin{bmatrix} 1 & -0.5 & 0 & 0 \\ -0.5 & 1 & -0.5 & 0 \\ 0 & -0.5 & 1 & -1/\sqrt{2} \\ 0 & 0 & -1/\sqrt{2} & 1 \end{bmatrix}. \tag{4.7}$$

4.1 Imposing Sparsity on the Connectivity Matrix

The minimization of the *sparsity of the weight matrix* keeps the number of its nonzero values to the minimum (Stanković, 2001; Stanković *et al.*, 2019c), thus resulting in graphs with the smallest possible number of edges.

Consider the vertex $n = 0$ and the graph signal observation vector as in (4.1), at this vertex. We can estimate the edge weights from this vertex to all other vertices, β_{0m}, $m = 1, 2, 3, \ldots, N-1$, by minimizing the cost function (Epskamp and Fried, 2018; Meinshausen *et al.*, 2006; Pourahmadi, 2011),

$$J_0 = \left\| \mathbf{y}_0 - \sum_{m=1}^{N-1} \beta_{0m} \mathbf{y}_m \right\|_2^2 + \rho \sum_{m=1}^{N-1} |\beta_{0m}|. \tag{4.8}$$

Physically, the first term promotes the correlation between the observations $\mathbf{y}_0$ at the considered vertex ($n = 0$) and the observations $\mathbf{y}_m$ at all the other vertices, for $m = 1, 2, 3, \ldots, N-1$; the second term promotes sparsity in the coefficient vector $\boldsymbol{\beta_0}$ (number of nonzero coefficients β_{0m}), while the parameter ρ balances between these two criteria.

The matrix form of the cost function (4.8) is given by

$$J_0 = \|\mathbf{y}_0^T - \mathbf{Y}_0^T \boldsymbol{\beta}_0^T\|_2^2 + \rho \|\boldsymbol{\beta}_0\|_1, \tag{4.9}$$

where $\mathbf{Y}_0$ is obtained from the matrix $\mathbf{X}_P$ after the first row is removed and used as $\mathbf{y}_0$, with

$$\boldsymbol{\beta}_0 = [\beta_{01}, \beta_{02}, \ldots, \beta_{0N-1}].$$

Example 6: For the correlation matrix from Example 5 and the observation vector, $\mathbf{y}_0$, at the vertex $n = 0$, given by

$$\mathbf{y}_0 = [x_1(0),\ x_2(0), \ldots, x_P(0)] = [\nu_0(1),\ \nu_0(2), \ldots, \nu_0(P)],$$

we can find the solution to (4.9) with $\rho = 0$, which corresponds to the two-norm minimization of the error function, given by

$$\frac{\partial J_0}{\partial \boldsymbol{\beta}_0^T} = 2\mathbf{Y}_0(\mathbf{y}_0^T - \mathbf{Y}_0^T \boldsymbol{\beta}_0^T) = \mathbf{0}$$

or

$$\boldsymbol{\beta}_0^T = (\mathbf{Y}_0 \mathbf{Y}_0^T)^{-1} \mathbf{Y}_0 \mathbf{y}_0^T = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 3 & 3 \\ 2 & 3 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \\ 0 \end{bmatrix},$$

since $\mathbf{Y}_0\mathbf{Y}_0^T$ and $\mathbf{Y}_0\mathbf{y}_0^T$ are submatrices of the correlation matrix $\mathbf{R}_x$, given in (4.5).

In the same way, the other three coefficient vectors, $\boldsymbol{\beta}_1^T$, $\boldsymbol{\beta}_2^T$, $\boldsymbol{\beta}_3^T$, were calculated to produce (with added zero-values at the diagonal) the coefficient matrix

$$\boldsymbol{\beta} = \begin{bmatrix} 0 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{4.10}$$

Since this procedure does not guarantee the symmetry of $\beta_{nm} = \beta_{mn}$, the edge weights could have also been calculated through the geometric mean,

$$W_{nm} = \sqrt{\beta_{nm}\beta_{mn}}, \tag{4.11}$$

to produce

$$\mathbf{W} = \begin{bmatrix} 0 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 1/\sqrt{2} \\ 0 & 0 & 1/\sqrt{2} & 0 \end{bmatrix}. \tag{4.12}$$

This weight matrix is symmetric and corresponds to an undirected graph.

The graph Laplacian, $\mathbf{L} = \mathbf{D} - \mathbf{W}$, is then obtained by changing the signs of the elements in $\mathbf{W}$ and adding appropriate diagonal elements, $\mathbf{D}$, such that the sum for each row or column is zero, that is

$$\mathbf{L} = \begin{bmatrix} 0.5 & -0.5 & 0 & 0 \\ -0.5 & 1 & -0.5 & 0 \\ 0 & -0.5 & 1.207 & -0.707 \\ 0 & 0 & -0.707 & 0.707 \end{bmatrix}.$$

Notice that the structure of nonzero off-diagonal elements in this matrix is the same as in the normalized precision matrix in (4.7), although the corresponding values were obtained through two quite different approaches to the estimation of the relations among graph data observed at different vertices.

LASSO approach. In general, the problem in (4.9) can be solved using the well established least absolute shrinkage and selection operator (LASSO) type minimization, the regression analysis method that

performs both variable selection and regularization, as

$$\boldsymbol{\beta}_0 = \text{lasso}(\mathbf{Y}_0^T, \mathbf{y}_0^T, \rho).$$

For more detail on the derivation and implementation of LASSO see Section 5 and Algorithm 1.

Algorithm 1. LASSO (ISTA variant), $\mathbf{B}$=lasso$(\mathbf{Y}, \mathbf{y}, \rho)$

Input:

- Observation column vector $\mathbf{y}$, $P \times 1$
- Observation matrix $\mathbf{Y}$, $P \times N$
- Sparsity promotion parameter ρ

1: $\mathbf{B} \leftarrow \mathbf{0}_{N\times 1}$
2: $\alpha \leftarrow 2\max\{\text{eig}(\mathbf{Y}^T\mathbf{Y})\}$
3: **repeat**
4: $\quad \mathbf{s} \leftarrow \dfrac{1}{\alpha}\mathbf{Y}^T(\mathbf{y} - \mathbf{YB}) + \mathbf{B}$
5: $\quad$ **for** $k \leftarrow 1$ to N **do**
6: $$B(k) \leftarrow \begin{cases} s(k) + \rho, & \text{for } s(k) < -\rho \\ 0, & \text{for } |s(k)| \leq \rho \\ s(k) - \rho, & \text{for } s(k) > \rho \end{cases}$$
7: **until** stopping criterion is satisfied

Output:

- Reconstructed coefficients $\mathbf{B}$

For the data from Example 6, the LASSO approach yields

$$\boldsymbol{\beta}_0 = \text{lasso}(\mathbf{Y}_0^T, \mathbf{y}_0^T, 0.01) = [0.49,\ 0,\ 0].$$

This result is almost the same as the first row (excluding the first element assumed to be zero) in the matrix $\boldsymbol{\beta}$ in (4.10), as was expected since the solution in the first row in (4.10) is already with maximum sparsity. Since in this setting the number of independent observations, P, could be significantly larger than the number of coefficients, β_{0m}, for this case the least squares estimation is optimal and there are no additional

degrees of freedom available to improve the sparsity of the solution (the solution, in this case is already with one nonzero element, that is, with the maximum possible sparsity). On the other hand, ways to promote sparsity would be necessary if the number of observations is smaller than the number of vertices (compressive sensing theory framework).

The minimization in (4.9) was performed for the vertex $n = 0$, and should be repeated for all vertices $n = 1, 2, \ldots, N-1$, through the cost function

$$J_n = \|\mathbf{y}_n^T - \mathbf{Y}_n^T \boldsymbol{\beta}_n\|_2^2 + \rho \|\boldsymbol{\beta}_n\|_1,$$

to obtain

$$\boldsymbol{\beta}_n = \text{lasso}(\mathbf{Y}_n^T, \mathbf{y}_n^T, \rho).$$

In general, if the resulting weight matrix, $\boldsymbol{\beta}$, is not symmetric then the edge weights could be calculated as $W_{nm} = \sqrt{\beta_{nm}\beta_{mn}}$, as mentioned in (4.11).

Example 7: As an example for graph learning from data using the LASSO approach, consider the graph from Figure 2.2, Part I and $P = 3{,}000$ observations, which were simulated by assuming external white Gaussian sources with zero-mean and variance $\sigma^2 = 1$, located at two randomly chosen vertices (see Section 5 and Figure 6.2). An $N \times P$ matrix of observed signal values, $\mathbf{X}_P$, was then formed, and from its rows the vector $\mathbf{y}_n$ and matrix $\mathbf{Y}_n$ were obtained. The matrix of coefficients $\boldsymbol{\beta} = [\beta_{mn}]_{N \times N}$ follows from $\text{lasso}(\mathbf{Y}_n^T, \mathbf{y}_n^T, \rho)$ with $n = 0, 1, 2, 3, 4, 5, 6, 7$ and $\rho = 0.2$, to yield

$$\boldsymbol{\beta} = \begin{bmatrix} 0 & 0.0 & 0.75 & 0.16 & 0 & 0 & 0 & 0 \\ 0.03 & 0 & 0.35 & 0 & 0.19 & 0 & 0 & 0.18 \\ 0.75 & 0.35 & 0 & 0.10 & 0.11 & 0 & 0 & 0 \\ 0.16 & 0 & 0.10 & 0 & 0 & 0 & 0.45 & 0 \\ 0 & 0.19 & 0.11 & 0 & 0 & 0.74 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.74 & 0 & 0 & 0.19 \\ 0 & 0 & 0 & 0.45 & 0 & 0 & 0 & 0.58 \\ 0 & 0 & 0 & 0 & 0 & 0.19 & 0.58 & 0 \end{bmatrix}.$$

The ground truth weights and the weights estimated through the LASSO are shown in Figures 4.1(a), (b). The estimation was repeated for the cases of (i) a smaller value of balance parameter $\rho = 0.05$ (reducing

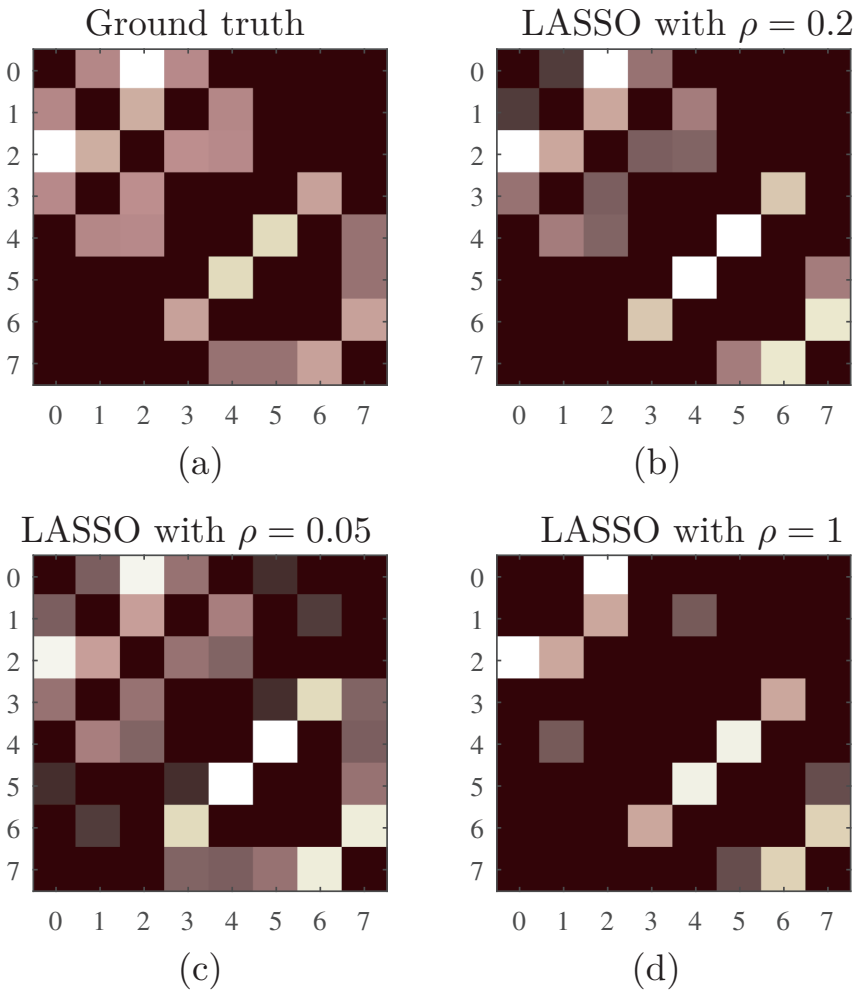


Figure 4.1: Estimation of the weight matrix for the graph from Figure 2.2 in Part I with color-coded element values. (a) Ground truth weight matrix. (b) Estimated weight matrix with LASSO and $\rho = 0.2$. (c) Estimated weight matrix with LASSO and $\rho = 0.05$. (d) Estimated weight matrix with LASSO and $\rho = 1$.

the sparsity contribution and resulting in an increased number of nonzero weights, as in Figure 4.1(c)), and (ii) a larger balance parameter $\rho = 1$ (strengthening the sparsity contribution and resulting in a reduced number of nonzero weights, as Figure 4.1(d)).

The same experiment was next repeated for the unweighted graph from Figure 2.1(a) in Part I, and the result is shown in Figure 4.2. In this case, the obtained values of $\boldsymbol{\beta}$ were used to decide whether $A_{mn} = 1$ or $A_{mn} = 0$.

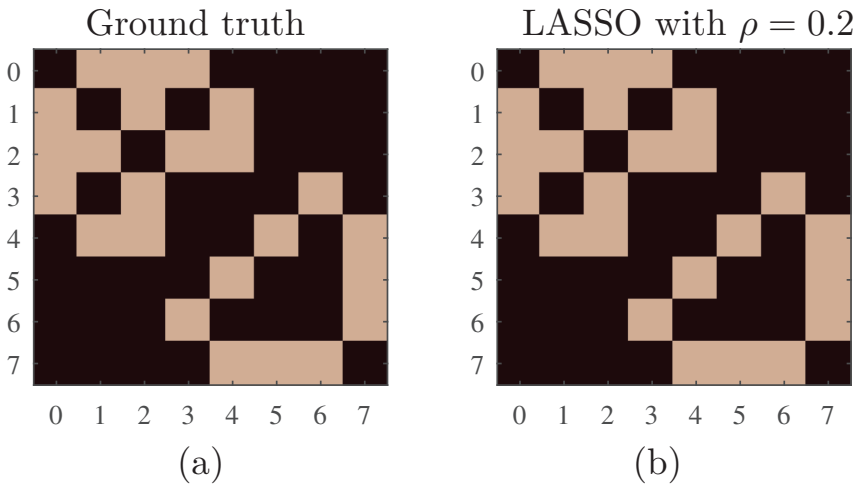


Figure 4.2: Adjacency matrix for the unweighted graph from Figure 2.1(a) in Part I. (a) Ground truth adjacency matrix. (b) Estimated adjacency matrix with LASSO and $\rho = 0.2$.

Example 8: The graph topology in the temperature estimation example in Part II, Section 2 was determined based on the geometry and geographic distances of the locations/vertices where the temperature is sensed (Stanković *et al.*, 2019b). Now, we shall revisit this example by simulating the temperature field, $\mathbf{X}$, at the locations shown in Figure 4.3(a) and over a period of time with the aim to learn the graph topology from this data. The simulated temperature field over $P = 150$ days is shown in Figure 4.3(b). The weight matrix calculated from the geographic positions of the vertices is denoted as the ground truth weight matrix, $\mathbf{W}$, and shown in Figure 4.3(c). The corresponding weight matrix, which is learned from data in Figure 4.3(b) using the column LASSO with $\rho = 0.2$, is given in Figure 4.3(d). Before the calculation of the correlation matrices, the mean value of the sensed temperatures was removed from $x_p(n)$, for each observation p.

4.2 Smoothness Constrained Learning of Graph Laplacian

Consider a set of noisy graph data, $x_p(n)$, measured over P observations, $p = 1, 2, \ldots, P$, at N vertices $n = 0, 1 \ldots, N-1$, of an undirected graph. The aim is to learn the graph connectivity (its graph Laplacian) from the observed data. To this end, it is necessary to find a signal, $y_p(n)$,

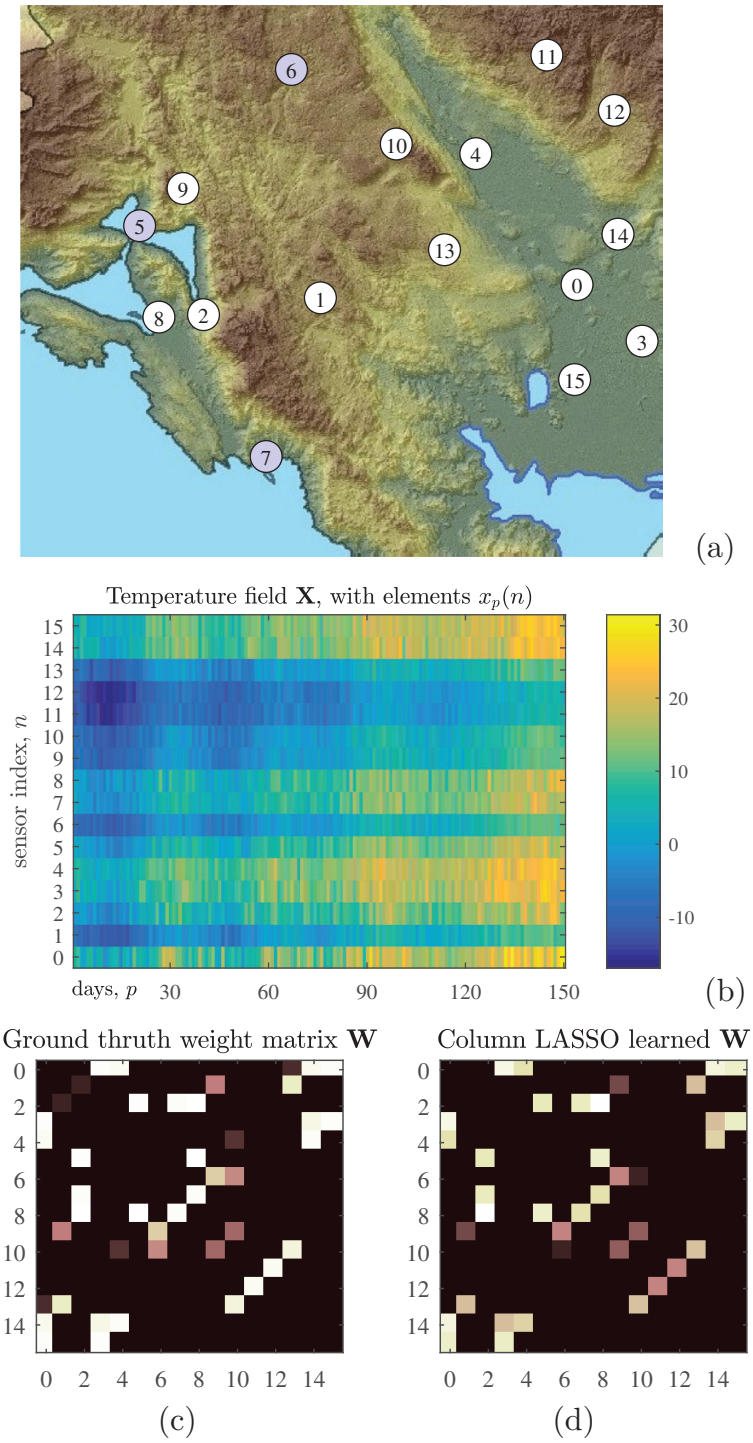


Figure 4.3: Data-based learning of graph topology in the temperature sensing example from Part II, Section 2. (a) Sensing locations in a geographic region along the Adriatic sea. (b) Temperatures measured at $N = 16$ sensing locations over $P = 150$ days. (c) Ground truth weight matrix, $\mathbf{W}$, obtained through geographic properties of the sensing locations as in Part II, Section 2. (d) The weight matrix, $\mathbf{W}$, estimated solely based on the analysis of data from (b) and using the LASSO approach.

that is close to the observations, $x_p(n)$, under the condition that $y_p(n)$ is as smooth as possible on a graph. This formulation is similar to that addressed in Part I.

Remark 3: *The smoothness condition* may be imposed based on the physically meaningful assumption that the data at close and strongly related vertices should have similar values, that is, without abrupt changes in signal values from vertex to vertex. This requirement imposes gradual change of data over the graph domain, as is the case in many practical applications (Chepuri *et al.*, 2017; Dong *et al.*, 2016, 2019; Kalofolias, 2016; Sadhanala *et al.*, 2016).

The graph signal, $y_p(n)$, can now be found by minimizing the cost function

$$J_p = \frac{1}{2}\|\mathbf{y}_p - \mathbf{x}_p\|_2^2 + \alpha \mathbf{y}_p^T \mathbf{L} \mathbf{y}_p, \quad \text{for } p = 1, 2, \ldots, P,$$

whereby the first term aims at finding $\mathbf{y}_p$ which is as close as possible to $\mathbf{x}_p$, while the second term, $\mathbf{y}_p^T \mathbf{L} \mathbf{y}_p$, promotes the smoothness of graph signal $\mathbf{y}_p$.

Remark 4: The difference in the problem considered here from the smoothing problem addressed in Part I is that here *the graph Laplacian (graph edges and their weights) is not known.* In other words, the graph Laplacian, $\mathbf{L}$, has to be determined along with the output signal $\mathbf{y}_p$, that is, the graph topology has to be learned from data.

Since we have available P graph-wise observations, we can form the $N \times P$ matrices

$$\mathbf{X}_P = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P]$$

and

$$\mathbf{Y}_P = [\mathbf{y}_1, \ \mathbf{y}_2, \ldots, \mathbf{y}_P].$$

Notice that here the vectors $\mathbf{y}_n$ above have to be calculated, and they are not related to the rearranged signal vectors, defined with the same notation, in the previous section.

4.3 Graph Topology Estimation with the Graph Laplacian Energy Condition

In addition to the signal smoothness, it is very useful to introduce the energy of graph Laplacian as an optimization condition, since *none of the above conditions is sensitive to the scaling of the graph Laplacian elements* and their possibly large values. Such cost function is then of the following form, Dong *et al.* (2016, 2019)

$$J = \sum_{p=1}^{P} \left[\frac{1}{2} \|\mathbf{y}_p - \mathbf{x}_p\|_2^2 + \alpha \mathbf{y}_p^T \mathbf{L} \mathbf{y}_p \right] + \beta \|\mathbf{L}\|_F^2,$$

where the penalty for the energy (squared Frobenius norm of a matrix) of the graph Laplacian, given by

$$\|\mathbf{L}\|_F^2 = \sum_m \sum_n L_{mn}^2$$

is involved in order to keep its values as low as possible.

The cost function for the whole set of P observations can now be written in a compact form as

$$J = \frac{1}{2} \|\mathbf{Y}_P - \mathbf{X}_P\|_F^2 + \alpha \text{Trace}\{\mathbf{Y}_P^T \mathbf{L} \mathbf{Y}_P\} + \beta \|\mathbf{L}\|_F^2, \tag{4.13}$$

where $\text{Trace}\{\mathbf{Y}_P^T \mathbf{L} \mathbf{Y}_P\}$ is a scalar which can be written as

$$\sum_{p=1}^{P} \mathbf{y}_p^T \mathbf{L} \mathbf{y}_p = \text{Trace}\{\mathbf{Y}_P^T \mathbf{L} \mathbf{Y}_P\}.$$

The above analysis assumes that the Laplacian has been first normalized. In order to avoid trivial solutions, the condition

$$\text{Trace}\{\mathbf{L}\} = N \tag{4.14}$$

is also used (as the diagonal elements of the ground truth normalized graph Laplacian are $L_{nn} = 1$), along with the condition that the off-diagonal elements are either zero or negative, that is

$$L_{mn} = L_{nm} \leq 0 \quad \text{for } n \neq m. \tag{4.15}$$

As with any Laplacian matrix, the sum of the graph Laplacian elements over every row or column is zero, that is

$$\sum_{m=0}^{N-1} L_{nm} = 0 \quad \text{and} \quad \sum_{n=0}^{N-1} L_{nm} = 0. \tag{4.16}$$

Remark 5: The optimization problem in (4.13) aims to learn the graph topology from the graph data by finding the Laplacian of a graph which is most likely to generate the observed graph data. The formulation in (4.13) is obviously jointly convex with respect to both the observed signal and the Laplacian, and can be solved through an iterative two-step procedure, given in Algorithm 2.

Algorithm 2. Iterative procedure for solving the problem of graph learning from data, given in (4.13)

1: Assume that

$$\mathbf{Y}_P = \mathbf{X}_P.$$

2: Estimate the graph Laplacian, $\mathbf{L}$, by minimizing

$$J_1 = \alpha \text{Trace}\{\mathbf{Y}_P^T \mathbf{L} \mathbf{Y}_P\} + \|\mathbf{L}\|_F^2$$

with the conditions given in (4.14)–(4.16), for the normalized graph Laplacian form.

3: For the Laplacian obtained in the Step 2, the signal $\mathbf{Y}_P$ is calculated by minimizing

$$J_2 = \frac{1}{2}\|\mathbf{Y}_P - \mathbf{X}_P\|_F^2 + \alpha \text{Trace}\{\mathbf{Y}_P^T \mathbf{L} \mathbf{Y}_P\}.$$

Iteratively repeat Step 2 and Step 3.
Step 3 has a closed form solution explained in Part I.

4.4 Learning of Generalized Laplacian-Graphical LASSO

The generalized Laplacian, $\mathbf{Q}$, is defined as Dong *et al.* (2016, 2019)

$$\mathbf{Q} = \alpha \mathbf{I} - \mathbf{N},$$

where $\mathbf{N}$ is a nonnegative symmetric matrix and $\mathbf{Q}$ is a symmetric positive semidefinite matrix. Any generalized Laplacian can be written as a sum of a standard Laplacian, $\mathbf{L}$, and a diagonal matrix, $\mathbf{P}$, that is

$$\mathbf{Q} = \mathbf{L} + \mathbf{P}.$$

Remark 6: The generalized Laplacian allows for the existence of self-loops on the vertices; these self-loops are defined by matrix $\mathbf{P}$.

Example 9: For the data in Example 5, the precision matrix is of the form

$$\mathbf{C} = \mathbf{R}_x^{-1} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}.$$

It may be considered as a generalized graph Laplacian since

$$\begin{aligned} \mathbf{R}_x^{-1} &= \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ &= \mathbf{L} + \mathbf{P}. \end{aligned}$$

This means that $\mathbf{R}_x^{-1}$ in this example may be interpreted as standard graph Laplacian with a self-loop at the vertex $n = 0$.

We will next show that owing to its physically relevant properties, the precision matrix, $\mathbf{C} = \mathbf{R}_x^{-1}$, can be used as an estimate of the generalized Laplacian, $\mathbf{Q}$.

Estimation of graph Laplacian through precision matrix. Consider a set of noisy signals $x_p(n)$ acquired over P observations, $p = 1, 2, \ldots, P$, at N vertices, $n = 0, 1 \ldots, N-1$, of an undirected graph. Our aim is to learn the graph connectivity (its Laplacian) based on the condition that the observed graph signal in the pth realization, $\mathbf{x}_p$, is as

smooth as possible on the graph defined by a generalized Laplacian, $\mathbf{Q}$, as explained in Remark 3. The cost function to achieve this goal can be conveniently defined by the signal smoothness function

$$J_p = \mathbf{x}_p^T \mathbf{Q} \mathbf{x}_p, \quad \text{for } p = 1, 2, \ldots, P.$$

The cumulative smoothness for all data $\mathbf{x}_p$, $p = 1, 2, \ldots, P$, is then expressed as

$$J = \frac{1}{P} \sum_{p=1}^{P} \mathbf{x}_p^T \mathbf{Q} \mathbf{x}_p, \tag{4.17}$$

while the correlation matrix of the all considered observations can be written as

$$\begin{aligned} \mathbf{R}_x &= \frac{1}{P} \sum_{p=1}^{P} \mathbf{x}_p \mathbf{x}_p^T \\ &= \frac{1}{P} [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P][\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P]^T \\ &= \frac{1}{P} \mathbf{X}_P \mathbf{X}_P^T. \end{aligned}$$

The smoothness index for all observations is now of the following form

$$J = \frac{1}{P} \sum_{p=1}^{P} \mathbf{x}_p^T \mathbf{Q} \mathbf{x}_p = \text{Trace}\{\mathbf{R}_x \mathbf{Q}\},$$

since

$$\begin{aligned} J &= \frac{1}{P} \sum_{p=1}^{P} \mathbf{x}_p^T \mathbf{Q} \mathbf{x}_p \\ &= \frac{1}{P} \text{Trace}\{[\mathbf{x}_1,\ \mathbf{x}_2, \ldots, \mathbf{x}_P]^T \mathbf{Q} [\mathbf{x}_1,\ \mathbf{x}_2, \ldots, \mathbf{x}_P]\} \\ &= \frac{1}{P} \text{Trace}\{\mathbf{X}_P^T \mathbf{Q} \mathbf{X}_P\} = \frac{1}{P} \text{Trace}\{\mathbf{X}_P \mathbf{X}_P^T \mathbf{Q}\} \\ &= \text{Trace}\{\mathbf{R}_x \mathbf{Q}\}. \end{aligned}$$

To avoid a trivial solution, the conditions for the generalized Laplacian should be incorporated. For symmetric positive definite matrices, all eigenvalues are positive, and since for any matrix, $\mathbf{Q}$, the product

of its eigenvalues is equal to $\det(\mathbf{Q})$, this condition can be included by adding the term $\ln(\det(\mathbf{Q}))$ to the cost function, to give

$$J = -\ln(\det(\mathbf{Q})) + \text{Trace}\{\mathbf{R}_x\mathbf{Q}\}. \tag{4.18}$$

Maximum likelihood interpretation. The interpretation of the cost function in (4.18) within the theory of Gaussian random signals and maximum likelihood estimation is given in Section 6.8. If we assume that the graph data at N vertices are N-dimensional random variables, with zero-mean and an unknown precision matrix $\mathbf{Q}$, then their N-dimensional probability density function is given by

$$P(\mathbf{x}_p) = \frac{1}{\sqrt{(2\pi)^p}}\sqrt{\det(\mathbf{Q})}\exp\left(-\frac{1}{2}\mathbf{x}_p^T\mathbf{Q}\mathbf{x}_p\right).$$

Within the maximum likelihood framework, the goal is to find the unknown parameter (matrix) $\mathbf{Q}$ so that the distribution fits the data in an optimal form. This optimal parameter matrix is obtained by differentiating the probability or its logarithm (log-likelihood) function,

$$\begin{aligned} -\ln\left\{P(\mathbf{x}_p)\sqrt{(2\pi)^p}\right\} &= -\ln\left\{\sqrt{\det(\mathbf{Q})}\exp\left(-\frac{1}{2}\mathbf{x}_p^T\mathbf{Q}\mathbf{x}_p\right)\right\} \\ &= -\frac{1}{2}\ln\{\det(\mathbf{Q})\} + \frac{1}{2}\mathbf{x}_p^T\mathbf{Q}\mathbf{x}_p, \end{aligned} \tag{4.19}$$

and setting the obtained derivative to zero.

Example 10: The concept of finding the best precision, Q, the reciprocal of the variance of Gaussian distribution, $Q = 1/\sigma^2$, to fit the data will be now illustrated on a simple setup. Assume that four observations of signal $x_p(n)$, $p = 1, 2, 3, 4$, at the vertex $n = 0$ are available, and are given by $x_1(0) = 0.2$, $x_2(0) = -0.3$, $x_3(0) = -0.4$, and $x_4(0) = -0.5$. It is also known that the data are zero-mean. The goal is to find the precision, $Q = 1/\sigma^2$, or variance, σ^2, of the Gaussian distribution of the observed data, given by

$$P(x_p(0)) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{x_p^2(0)}{2\sigma^2}\right) = \sqrt{\frac{Q}{2\pi}}\exp\left(-\frac{1}{2}x_p(0)Qx_p(0)\right)$$

which corresponds to the best fit to the observed data. The log-likelihood function of the joint distribution of these four observed data points is

then

$$\begin{aligned} J &= -\ln(P(x_1(0))P(x_2(0))P(x_3(0))P(x_4(0))) \\ &= -\ln\left(\frac{1}{4\pi^2}Q^2 e^{-\frac{1}{2}0.2^2 Q} e^{-\frac{1}{2}0.3^2 Q} e^{-\frac{1}{2}0.4^2 Q} e^{-\frac{1}{2}0.5^2 Q}\right) \\ &= 2\ln(2\pi) - 2\ln(Q) + \frac{1}{2}(0.2^2 + 0.3^2 + 0.4^2 + 0.5^2)Q \\ &= 2\ln(2\pi) - 2\ln(Q) + \frac{1}{2}0.54Q. \end{aligned}$$

The differentiation of this expression with respect to $Q = 1/\sigma^2$ produces $-2/Q + \frac{1}{2}0.54 = 0$ or $Q = 4/0.54 = 7.4$ and

$$\sigma = \sqrt{1/Q} = 0.36.$$

The same value would have been produced by a simple standard deviation estimator, $\sigma = \sqrt{(0.2^2 + 0.3^2 + 0.4^2 + 0.5^2)/4}$.

Example 11: Similar analysis, as in the previous example, can be performed for P observations at two vertices, $n = 0$ and $n = 1$, $[x_p(0), x_p(1)]^T$. The goal is to estimate the parameters of the precision matrix

$$\mathbf{Q} = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}$$

of the joint Gaussian distribution of $[x_p(0), x_p(1)]^T$, defined as

$$P([x_p(0), x_p(1)]^T) = \frac{\sqrt{\det(\mathbf{Q})}}{2\pi} e^{-\frac{1}{2}[x_p(0), x_p(1)]\mathbf{Q}[x_p(0), x_p(1)]^T} \tag{4.20}$$

$$= \frac{\sqrt{Q_{11}Q_{22} - Q_{12}Q_{21}}}{2\pi} \times e^{-\frac{1}{2}(Q_{11}x_p^2(0) + (Q_{12}+Q_{21})x_p(0)x_p(1) + Q_{22}x_p^2(1))}. \tag{4.21}$$

Using P available realizations,

$$[x_1(0), x_1(1)], [x_2(0), x_2(1)], \ldots, [x_P(0), x_P(1)]$$

and the corresponding P-variate normal distribution of two variables as a product of P distributions as in (4.21), we can find the parameters $Q_{11}, Q_{12}, Q_{21}, Q_{22}$ which produce the best distribution fit, using the partial derivatives of the log-likelihood function.

For example, a partial derivative of the log-likelihood function with respect to Q_{11} would produce

$$-\frac{P}{2}\frac{Q_{22}}{\sqrt{Q_{11}Q_{22}-Q_{12}Q_{21}}}+\frac{1}{2}(x_1^2(0)+x_2^2(0)+\cdots+x_P^2(0))=0.$$

Observe that the term

$$\frac{Q_{22}}{\sqrt{Q_{11}Q_{22}-Q_{12}Q_{21}}}=\frac{Q_{22}}{\sqrt{\det(\mathbf{Q})}}$$

is just the first element of the inverse of matrix $\mathbf{Q}$, while the term $(x_1^2(0)+x_2^2(0)+\cdots+x_P^2(0))$ is the first element of the correlation matrix $\mathbf{R}_x$, multiplied by P. In a similar way, the derivations over Q_{12}, Q_{21}, and Q_{22}, will produce the remaining elements of the inverse of matrix $\mathbf{Q}$ and the correlation matrix $\mathbf{R}_x$. In the matrix notation, the solution to the so obtained system of the four equations is given by

$$\mathbf{Q}^{-1}=\frac{1}{P}\begin{bmatrix}\sum_{p=1}^{P}x_p^2(0) & \sum_{p=1}^{P}x_p(0)x_p(1)\\ \sum_{p=1}^{P}x_p(1)x_p(0) & \sum_{p=1}^{P}x_p^2(1)\end{bmatrix}=\mathbf{R}_x.$$

Notice that at least $P=2$ independent observations, $P\geq N$, are needed, since for $P=1$ observation, $P<N$, and the rank of the correlation matrix, $\mathbf{R}_x$, would be 1, which is lower than its dimension. In that case, the correlation matrix would not be invertible.

The cost function in (4.18) minimizes the logarithm of the joint probability density function of a graph signal $\mathbf{x}_p$ under the Gaussian assumption. The minimization of the cost function J with respect to $\mathbf{Q}$, with $\partial J/\partial\mathbf{Q}=\mathbf{0}$, produces

$$\frac{\partial J}{\partial\mathbf{Q}}=\frac{\partial}{\partial\mathbf{Q}}(-\ln(\det(\mathbf{Q}))+\text{Trace}\{\mathbf{R}_x\mathbf{Q}\}). \tag{4.22}$$

In order to find this derivative, we will use the relation among the trace of a positive semidefinite matrix, its eigenvalues, λ_k, and the trace of the eigenvalue matrix, $\mathbf{\Lambda}$, in the form

$$\begin{aligned}\ln(\det(\mathbf{Q}))&=\sum_{k=1}^{N}\ln(\lambda_k)\\&=\text{Trace}(\ln(\mathbf{\Lambda}))=\text{Trace}(\ln(\mathbf{Q})).\end{aligned} \tag{4.23}$$

Note also that for a differentiable matrix function, $f(\mathbf{Q})$, the following holds

$$\frac{\partial}{\partial \mathbf{Q}}(\text{Trace}\{f(\mathbf{Q})\}) = \frac{\partial f(\mathbf{Q})}{\partial \mathbf{Q}}. \tag{4.24}$$

Having in mind the properties in (4.23) and (4.24), we can write

$$\frac{\partial J}{\partial \mathbf{Q}} = -\mathbf{Q}^{-1} + \mathbf{R}_x. \tag{4.25}$$

The best estimate of $\mathbf{Q}$ follows from $\partial J/\partial \mathbf{Q} = \mathbf{0}$, and has the form

$$\mathbf{Q} = \mathbf{R}_x^{-1}. \tag{4.26}$$

Remark 7: The solution in (4.26), being equal to the precision matrix, can be used as the generalized Laplacian estimate in order to obtain the underlying graph structure.

Example 12: The weight matrix which corresponds to the inverse of the correlation matrix $\mathbf{R}_x$, for which the positive and small off-diagonal values were set to zero, is shown in Figure 4.4(b). Here, we consider the graph from Figure 2.2 in Part I and $P = 10{,}000$ observations. The observations were simulated by assuming white Gaussian external sources with zero-mean and variance $\sigma^2 = 1$, located at a randomly chosen vertex (as described in more detail in Section 4).

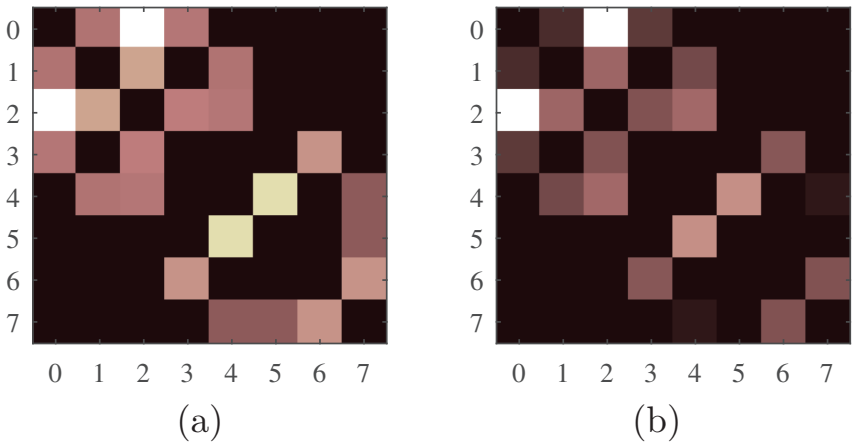


Figure 4.4: Weight matrix for the graph from Figure 2.2 in Part I. (a) Ground truth weight matrix. (b) Estimated weight matrix using the inverse correlation (precision) matrix.

Remark 8: Notice that the correlation matrix, $\mathbf{R}_x$, may be singular. The correlation matrix, $\mathbf{R}_x$, is *always singular* when the number of observations, P, is lower than the number vertices (dimension of the correlation matrix, N) that is, $N > P$. This follows from the fact that the correlation matrix is formed as a combination of P signals, $\mathbf{R}_x = \frac{1}{P}\sum_{p=1}^{P} \mathbf{x}_p \mathbf{x}_p^T$, which means that its dimensionality is spanned over at most P independent vectors (eigenvectors), and therefore its rank is equal to or lower than P (see Example 19 in Section 5).

Also, this form will not produce a matrix satisfying the conditions for a generalized Laplacian. The inverse correlation function may also have positive off-diagonal values. Therefore, for a reliable solution, the cost function in (4.18) should have additional constraints. Here, we will present two of such constraints.

Graphical LASSO. In this approach, the classical reconstruction formulation of a sparse signal is used as the additional constraint on the precision matrix and the cost function from (4.18) (Friedman *et al.*, 2008). The sparsity constraint on the generalized Laplacian is added to achieve the solution with the smallest possible number of nonzero entries in the estimated graph weight matrix – the smallest number of edges. The sparsity condition also allows for the problem solution with a reduced correlation matrix rank (as within the compressive sensing framework described in Part II). The cost function, with the included sparsity penalty function, $\|\mathbf{Q}\|_1$, is then defined as

$$J = -\ln(\det(\mathbf{Q})) + \text{Trace}\{\mathbf{R}_x\mathbf{Q}\} + \rho\|\mathbf{Q}\|_1. \tag{4.27}$$

This minimization problem can be solved in many ways, one of which is the graphical LASSO algorithm, an extension of the standard LASSO algorithm to graph problems (see Algorithm 3 for the implementation and Section 5 for the derivation of graphical LASSO).

Example 13: For the same signal as in Example 12, the weight matrix obtained using the graphical LASSO,

$$\mathbf{W} = \text{glasso}(\mathbf{R}_x, 0.3),$$

where both positive and small element values are set to zero, is shown in Figure 4.5(b) (see also Example 19).

Algorithm 3. Graphical LASSO, $\mathbf{Q} = \text{glasso}(\mathbf{R}, \rho)$

Input:

- Correlation matrix $\mathbf{R}$
- Regularization parameter ρ

$M_i \leftarrow 100$, $E_p \leftarrow 0.0001$
$[p, n] \leftarrow \text{size}(\mathbf{R})$
$C_p \leftarrow \text{mean}(|\mathbf{R} - \text{diag}(\text{diag}(\mathbf{R}))|)E_p$
$\mathbf{V}_0 = \mathbf{R} + \rho\mathbf{I}$
$\mathbf{V} = \mathbf{V}_0$
for $r = 1$ to M_i **do**
    for $j = p$ to 1 step -1 **do**
        $\mathbf{V}_{11} \leftarrow \mathbf{V}$
        $\mathbf{V}_{11} \leftarrow \mathbf{V}_{11}$ with removed jth row
        $\mathbf{V}_{11} \leftarrow \mathbf{V}_{11}$ with removed jth column
        $v_{22} \leftarrow V(j, j)$
        $\mathbf{r}_{12} \leftarrow j$th column of $\mathbf{R}$
        $\mathbf{r}_{12} \leftarrow \mathbf{r}_{12}$ with removed jth element
        $\mathbf{A} \leftarrow \sqrt{\mathbf{V}_{11}}$
        $\mathbf{b} \leftarrow (\sqrt{\mathbf{V}_{11}})^{-1}\mathbf{r}_{12}$
        $\boldsymbol{\beta} = \text{lasso}(\mathbf{A}, \mathbf{b}, \rho)$, as in Algorithm 1
        $\mathbf{v}_{12} \leftarrow \mathbf{V}_{11}\boldsymbol{\beta}$
        $\mathbf{V} \leftarrow \mathbf{V}$ with $\mathbf{v}_{12}$ inserted as the jth column
        $\mathbf{v}_{12} \leftarrow \mathbf{v}_{12}^T$ with v_{22} inserted as the jth element
        $\mathbf{V} \leftarrow \mathbf{V}$ with $\mathbf{v}_{12}$ inserted as the jth row
    if $\text{mean}(|\mathbf{V} - \mathbf{V}_0| < C_p)$ **break**, **end**
    $\mathbf{V}_0 = \mathbf{V}$
$\mathbf{Q} = \mathbf{V}^{-1}$

- Estimated precision matrix $\mathbf{Q}$

Generalized Laplacian constrained approach. Another possible approach employs the Lagrange multipliers, $\mathbf{B}$, which are added in such a way that these values do not change the diagonal elements of $\mathbf{Q}$, and

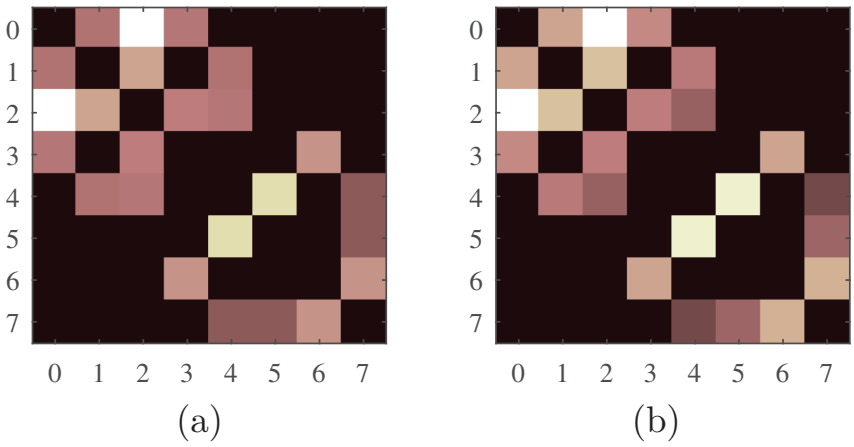


Figure 4.5: Weight matrix for the graph from Figure 2.2 in Part I. (a) Ground truth weight matrix. (b) Estimated weight matrix using the graphical LASSO and inverse correlation (precision) matrix.

ensure that all

$$Q_{mn} = Q_{nm} \leq 0$$

for $n \neq m$, with $B_{nm} = B_{mn} \geq 0$. The diagonal elements of matrix $\mathbf{B}$ are therefore $B_{nn} = 0$. Finally, the condition $B_{nm}Q_{nm} = 0$ for all n and m is used. In this case, the minimization solution for the generalized Laplacian is obtained as

$$\mathbf{Q} = (\mathbf{R}_x + \mathbf{B})^{-1}$$

based on the cost function

$$J = -\ln(\det(\mathbf{Q})) + \text{Trace}\{\mathbf{R}_x\mathbf{Q}\} + \text{Trace}\{\mathbf{B}\mathbf{Q}\}.$$

The results obtained in this case are similar to those obtained with the graphical LASSO approach.

4.5 Graph Topology Learning Based on the Eigenvectors

Assume that the available observations of a graph signal, $x_p(n)$, are graph wide sense stationary (GWSS), that is, they can be considered as the output of a linear system $H(\mathbf{L})$, driven by white noise, ε_p, as the input. In other words, the signal on a graph is formed using a linear combination of a white noise realization, ε_p, and its graph shifted versions. The output signal after M such graph shifts, defined by the

normalized Laplacian, is given by

$$\mathbf{x}_p = (h_M \mathbf{L}^M + h_{M-1}\mathbf{L}^{M-1} + \cdots + h_1 \mathbf{L}^1 + h_0 \mathbf{L}^0)\boldsymbol{\varepsilon}_p. \tag{4.28}$$

This graph signal can be written in a compact form

$$\mathbf{x}_p = H(\mathbf{L})\boldsymbol{\varepsilon}_p,$$

with its correlation matrix given by (for $\sigma_\varepsilon^2 = 1$)

$$\begin{aligned}\mathbf{R}_x &= \frac{1}{P}\sum_{p=1}^{P} \mathbf{x}_p \mathbf{x}_p^T = \frac{1}{P}\sum_{p=1}^{P} H(\mathbf{L})\boldsymbol{\varepsilon}_p \boldsymbol{\varepsilon}_p^T H^T(\mathbf{L}) \\ &= H(\mathbf{L})\left(\frac{1}{P}\sum_{p=1}^{P} \boldsymbol{\varepsilon}_p \boldsymbol{\varepsilon}_p^T\right) H^T(\mathbf{L}) \\ &= H(\mathbf{L})H^T(\mathbf{L}) = \mathbf{U}^T |H(\boldsymbol{\Lambda})|^2 \mathbf{U} \end{aligned} \tag{4.29}$$

where $\boldsymbol{\varepsilon}_p$ is a white noise with unit variance, and $\mathbf{U}$ is the matrix of graph Laplacian eigenvectors, $\mathbf{L} = \mathbf{U}^T \boldsymbol{\Lambda} \mathbf{U}$.

From (4.29), it is now obvious that we can learn about the graph eigenvectors from the decomposition of the autocorrelation matrix. The same holds for the precision matrix, $\mathbf{Q} = \mathbf{R}_x^{-1}$, since the inverse matrix has the same eigenvectors as the original matrix.

For the normalized graph Laplacian, it is straightforward to relate the Laplacian, $\mathbf{L}_N$, based shift and the normalized weight matrix, $\mathbf{W}_N$, based shift since

$$\mathbf{L}_N^p = (\mathbf{I} - \mathbf{W}_N)^p = \mathbf{I} - p\mathbf{W}_N + \cdots + (-1)^p \mathbf{W}_N^p.$$

Therefore from (4.29), in order to estimate the graph connectivity (estimate its Laplacian or adjacency matrix) we can use the eigenvectors of the autocorrelation matrix.

Remark 9: Since we do not know $H(\boldsymbol{\Lambda})$, it is natural to assume that the graph is defined by the eigenvalues, $\boldsymbol{\Lambda}$, that produce the smallest number of edges. This can be achieved by minimizing the number of nonzero values in $\mathbf{L}$ for the given eigenvectors (Marques *et al.*, 2017; Segarra *et al.*, 2017).

The minimization problem of determining a graph now becomes

$$\min_{\lambda_k} \|\mathbf{L}\|_0 \text{ subject to } \mathbf{L} = \sum_{k=0}^{N-1} \lambda_k \mathbf{u}_k \mathbf{u}_k^T, \tag{4.30}$$

while the convex (norm-one) form of this minimization problem is

$$\min_{\lambda_k} \|\mathbf{L}\|_1 \text{ subject to } \mathbf{L} = \sum_{k=0}^{N-1} \lambda_k \mathbf{u}_k \mathbf{u}_k^T. \tag{4.31}$$

Remark 10: The convex norm-one based form in (4.31) can produce the same solution as the original norm-zero form in (4.30) if the Laplacian sparsity is low and the Laplacian satisfies some other conditions (in the sense discussed in Section 4.2).

Since the eigenvectors are obtained from the decomposition of the correlation matrix, spectral analysis performed in this way is related to principal component analysis (PCA), where the signal is decomposed through the set of eigenvectors of the correlation matrix.

This approach to graph topology learning can be summarized through the following steps.

1. For a given set of graph signal observations, $\mathbf{x}_p$, $p = 1, 2, \ldots, P$, calculate the correlation matrix

$$\mathbf{R}_x = \frac{1}{P} \sum_{p=1}^{P} \mathbf{x}_p \mathbf{x}_p^T. \tag{4.32}$$

2. Perform the eigen decomposition of the correlation matrix, in the form

$$\begin{aligned} \mathbf{R}_x &= \mathbf{U}^T \mathbf{\Lambda}_{R_x} \mathbf{U} \\ \mathbf{\Lambda}_{R_x} &= \mathbf{U} \mathbf{R}_x \mathbf{U}^T. \end{aligned} \tag{4.33}$$

3. Find the eigenvalues, λ_k, of the graph Laplacian, $\mathbf{L} = \mathbf{U}^T \mathbf{\Lambda} \mathbf{U}$, such that it assumes the sparsest possible form, using the minimization

$$\min_{\lambda_k} \|\mathbf{L}\|_1 \text{ subject to } \mathbf{L} = \sum_{k=0}^{N-1} \lambda_k \mathbf{u}_k \mathbf{u}_k^T. \tag{4.34}$$

Dimensionality-reduction methods. It is often reasonable to assume that the observed graph signals are generated by exciting a low-order graph system with white noise as the input. However, the problem of estimating the polynomial coefficients from its samples at unknown

(eigenvalue) positions is under-determined and cannot be directly solved. By adding the constraint that true eigenvalue positions should produce a sparse graph Laplacian, the solution becomes tractable within the compressive sensing framework (Stanković *et al.*, 2020).

In this way, instead of the minimization over N variables, λ_k, $k = 0, 1, \ldots, N-1$, we can find the Laplacian eigenvalues starting from the eigendecomposition of the correlation matrix of a signal produced by a system on a graph, that is,

$$\mathbf{R}_x = \mathbf{U}|H(\mathbf{\Lambda})|^2\mathbf{U}^T = \mathbf{U}\mathbf{\Lambda}_{R_x}\mathbf{U}^T. \tag{4.35}$$

Assume that the transfer function of the graph system is of a polynomial form

$$H(\lambda_k) = h_0 + h_1\lambda_k + h_2\lambda_k^2 + \cdots + h_M\lambda_k^M \tag{4.36}$$

with $M \ll N$. From the correlation matrix eigendecomposition in (4.35), we have N values of $H(\lambda_k)$ obtained as square roots of the eigenvalues of the correlation matrix, $\lambda_k^{(\mathbf{R}_x)}$. Without loss of generality, we will assume a nondecreasing $H(\lambda_k)$, that is $H(\lambda_{k-1}) \leq H(\lambda_k)$. The problem now boils down to the determination of the Laplacian eigenvalues, λ_k, $k = 0, 1, \ldots, N-1$, having in mind that $\lambda_0 = 0$, $\sum_{k=0}^{N-1} \lambda_k = N$ and that there exist (unknown) coefficients h_i, $i = 0, 1, \ldots, M$ such that (4.36) is satisfied for each k, while the true values λ_k produce the sparsest graph Laplacian, $\mathbf{L}$.

The estimation of the system coefficients, Laplacian eigenvalues and Laplacian itself is performed using this **polynomial fitting method** in the following way.

1. Select $(M+1)$ indices $m_0 = 0 < m_1 < \cdots < m_M = N$ with the corresponding transfer function values $H(\lambda_{m_i})$, for $i = 0, 1, \ldots, M$. Assume that $(M+1)$ eigenvalues are $\bar{\lambda}_0 = 0$, $\bar{\lambda}_{m_1} = \xi_1$, $\bar{\lambda}_{m_2} = \xi_2, \ldots, \bar{\lambda}_{m_{M-1}} = \xi_{M-1}$, $\bar{\lambda}_{m_M} = 1$, where $0 < \xi_1 < \xi_2 < \cdots < \xi_{M-1} < 1$.

2. Then, the coefficients of an Mth order polynomial

$$P(\bar{\lambda}) = a_0 + a_1\bar{\lambda} + a_2\bar{\lambda}^2 + \cdots + a_M\bar{\lambda}^M$$

can be found such that $P(\hat{\lambda}_i) = H(\lambda_{m_i})$, for $i = 0, 1, \ldots, M$, is a Lagrange polynomial of Mth order defined by $(M+1)$ points.

3. Now, the eigenvalues $\bar{\lambda}_k$, for each k, can be calculated as a solution of
$$P(\bar{\lambda}) = H(\lambda_k), \quad 0 \leq \bar{\lambda} \leq 1$$
for the unknown $\bar{\lambda}$. Note that this solution is unique if the polynomial $P(\hat{\lambda})$ is an increasing function for $0 \leq \hat{\lambda} \leq 1$.

4. Having in mind that $\sum_{k=0}^{N-1} \lambda_k = N$, the eigenvalues, $\hat{\lambda}_k$, can be found by scaling the obtained values, $\bar{\lambda}_k$, for each k, as $\hat{\lambda}_k = N\bar{\lambda}_k / \sum_{k=0}^{N-1} \bar{\lambda}_k$.

5. For the so obtained estimates of the eigenvalues, $\hat{\lambda}_k$, the normalized graph Laplacian can be calculated as $\mathbf{L} = \mathbf{U}\hat{\mathbf{\Lambda}}\mathbf{U}^T$, where $\hat{\mathbf{\Lambda}}$ is a diagonal matrix with $\hat{\lambda}_k$ on the diagonal.

6. The above procedure should be repeated for various $0 < \xi_1 < \xi_2 < \cdots < \xi_{M-1} < 1$ and the final solution is obtained by minimizing the energy normalized sparsity condition, given by
$$\min_{\xi_1, \xi_2, \ldots, \xi_{M-1}} \frac{\|\mathbf{L}\|_1}{\sqrt{\|\mathbf{L}\|_2}}.$$

Notice that for $M = 1$, we should consider only two points in Step 1, and there is no need for the minimization of variables ξ_i. For $M = 2$, we have one minimization variable $0 < \xi_1 < 1$. For $M = 3$, the minimization is performed over only two variables, $0 < \xi_1 < \xi_2 < 1$. Given that dimensionality of the minimization problem is $(M - 1)$, and since $M \ll N$, the dimensionality reduction of this method when compared to (4.34) is evident.

The spectral indices $0 = m_0, m_1, \ldots, m_M = N$, selected in Step 1, should be equally spaced over N possible values. For $M = 2$, the index m_1 should be close to $(N - 1)/2$, while for $M = 3$ the indices m_1 and m_2 should be close to $(N - 1)/3$ and $2(N - 1)/3$, respectively.

Example 14: Consider a graph with $N = 8$ vertices, for which the weight matrix is given in Figure 4.6(a). An $N \times P$ matrix of the simulated signal, $\mathbf{X}_P$, was formed by calculating the graph signal as in (4.28), with a given graph, its weight matrix, $\mathbf{W} = \mathbf{I} - \mathbf{L}$, the normalized Laplacian, $\mathbf{L}$, system order M, and system coefficients, $h_0, h_1, \ldots, h_M$.

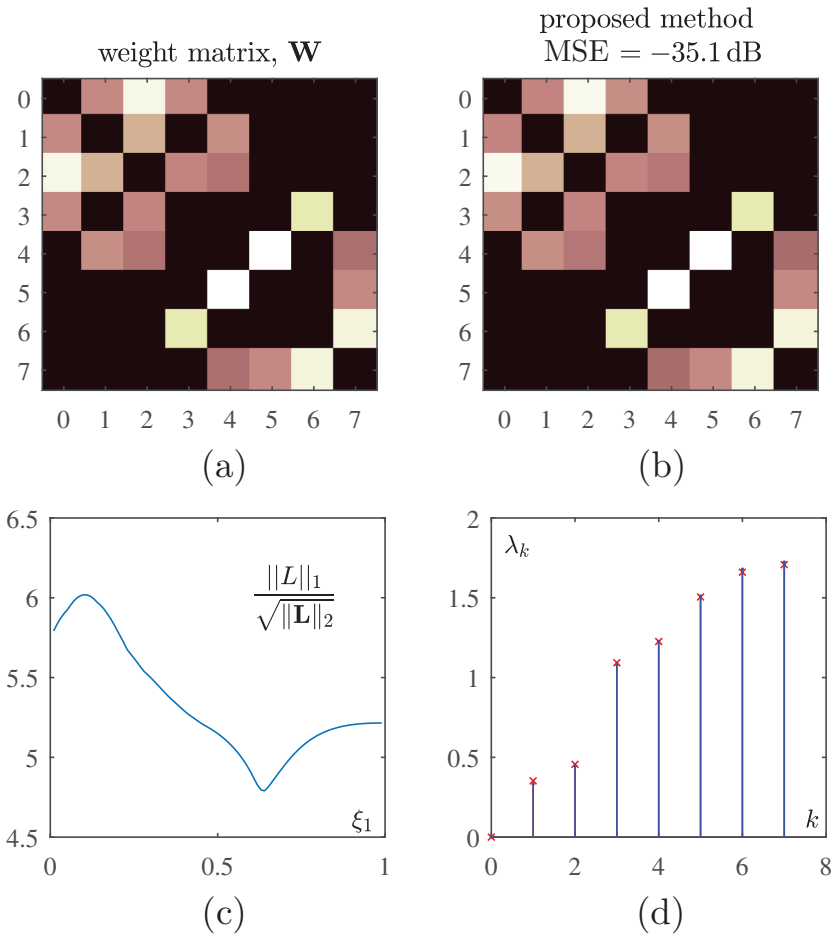


Figure 4.6: Estimation of the weight matrix, $\mathbf{W} = \mathbf{I} - \mathbf{L}$, for the graph with $N = 8$ vertices. (a) Ground truth weight matrix. (b) Estimated weight matrix using sparsity minimization of the normalized Laplacian. (c) Sparsity measure minimization, as a function of parameter ξ_1. (d) The exact (blue lines) and estimated (red crosses) eigenvalues of the normalized Laplacian.

White Gaussian external sources, $\boldsymbol{\varepsilon}_p$, with zero-mean and variance $\sigma^2 = 1$ were assumed in all $P = 10{,}000$ realizations.

The presented polynomial fitting method was implemented for the assumed degree $M = 2$ of the polynomial $H(\lambda)$, with $h_0 = 0.3$, $h_1 = 0.2$, and $h_2 = 0.5$ used in the graph signal simulation, according to (4.28). By forming $\mathbf{R}_x$ from $\mathbf{X}_P$ and after its eigendecomposition, the eigenvectors $\mathbf{U}$ were estimated, while the eigenvalues of the correlation matrix were used to calculate $H(\lambda_k) = \sqrt{\lambda_k^{(\mathbf{R}_x)}}$.

Observe that the polynomial fitting method reduces to the one-dimensional minimization over variable $0 < \xi_1 < 1$, as shown in Figure 4.7. After the minimum value of the sparsity measure is found,

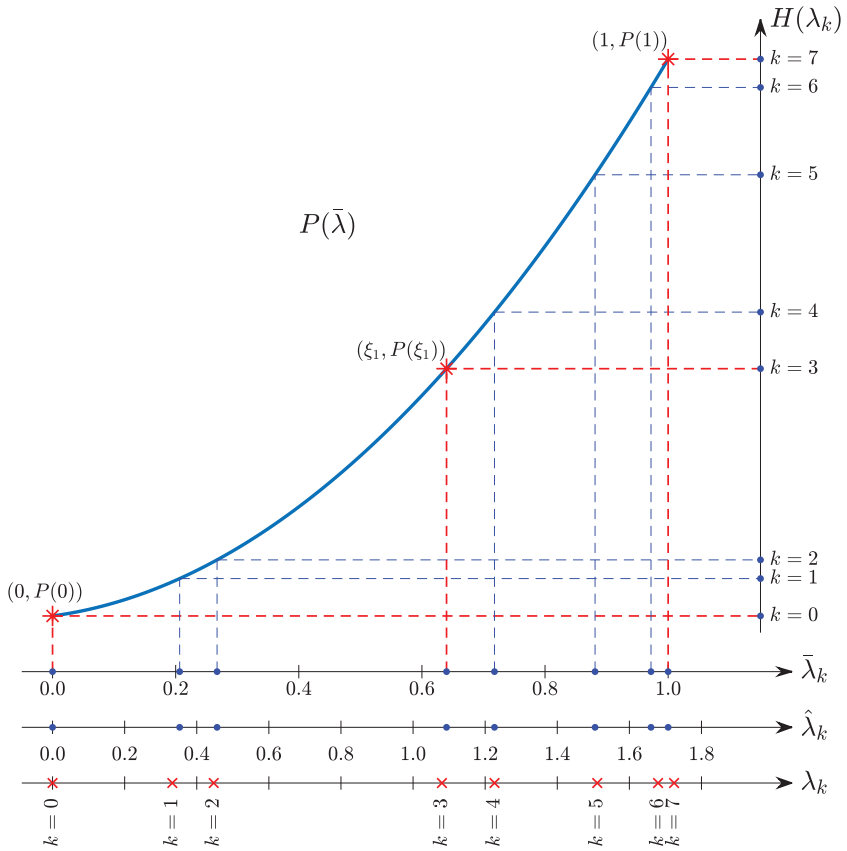


Figure 4.7: Illustration of eigenvalue calculation based on their second order polynomial, obtained from $H(\lambda_k) = \sqrt{\lambda_k^{(\mathbf{R}_x)}}$.

the eigenvalues are calculated with the corresponding parameter, ξ_1. The Laplacian then follows from $\mathbf{L} = \mathbf{U}^T \mathbf{\Lambda} \mathbf{U}$.

In this case, the obtained error in the weight matrix elements (absolute value of the off-diagonal elements of the Laplacian) is characterized by the MSE $= -35.1$ dB, with the results presented in Figure 4.6. The true weight matrix, $\mathbf{W} = \mathbf{I} - \mathbf{L}$, along with the estimated one, is given in Figures 4.6(a) and (b), and the sparsity measure function is plotted in Figure 4.6(c), while the true and the estimated Laplacian eigenvalues are given in Figure 4.6(d).

Example 15: The experiment from Example 14 was repeated for a low number of observations, $P = 8N_L = 256$, where $N_L = 32$ is the sparsity of the Laplacian matrix according to practical hints for the number of measurements and sparsity (Candès *et al.*, 2006). The reconstruction using the polynomial fitting produced the MSE $= -18.0$ dB.

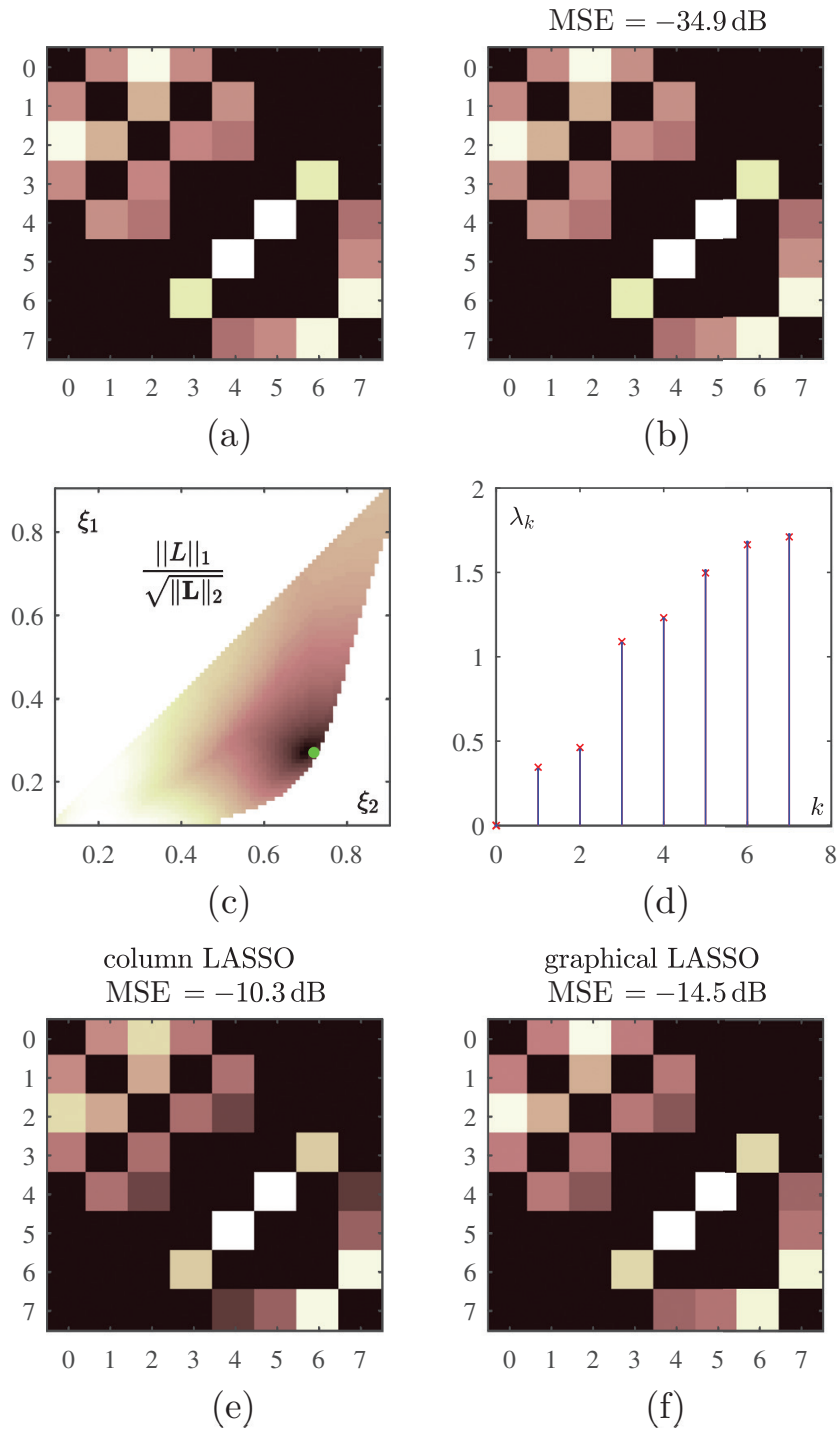


Figure 4.8: Estimation of the weight matrix for the graph with $N = 8$ vertices. (a) Ground truth weight matrix. (b) Estimated weight matrix using the polynomial fitting method. (c) Sparsity measure minimization, as a function of parameters ξ_1 and ξ_2. (d) The exact (blue lines) and estimated (red crosses) eigenvalues of the normalized Laplacian. (e) Estimated weight matrix using the LASSO minimization. (f) Estimated weight matrix using the graphical LASSO.

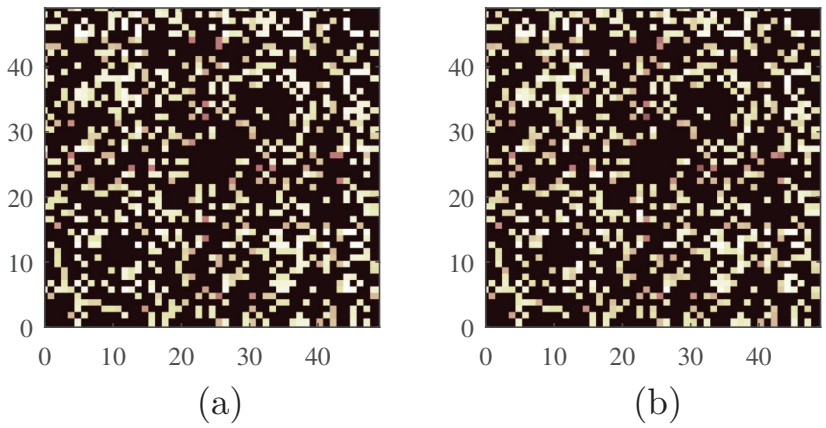


Figure 4.9: Estimation of the weight matrix for a graph with $N = 50$ randomly positioned vertices. (a) Ground truth weight matrix. (b) Estimated weight matrix using sparsity minimization of the normalized Laplacian and the polynomial fitting method.

In this experiment, we assumed $M = 3$ and $h_0 = 0.4$, $h_1 = 0.5$, $h_2 = 0.4$, and $h_3 = 0.2$ when simulating the graph signal, $\mathbf{X}_P$. The correlation matrix was estimated using this simulated signal, along with its eigenvectors and eigenvalues. We now have two minimization variables ξ_1 and ξ_2, $0 < \xi_1 < \xi_2 < 1$. The results for the polynomial fitting method are presented in Figures 4.8(a)–(d). The obtained estimation error was MSE $= -34.9$ dB. The sparsity measure function (Figure 4.8(c)) is now two-dimensional and is calculated only when unique solutions are obtained in Step 3 of the polynomial fitting method. These results were compared with those obtained using the rows of the correlation matrix, $\boldsymbol{\beta}_n = \text{lasso}(\mathbf{Y}_n^T, \mathbf{y}_n^T, 0.2)$ (Figure 4.8(e)) and graphical LASSO, $\mathbf{Q} = \text{glasso}(\mathbf{R}_x, 0.3)$ (Figure 4.8(f)), with the optimized values of the parameter ρ. In these cases, the obtained error in the weight matrix elements was characterized by MSE $= -10.3$ dB and MSE $= -14.5$ dB, respectively.

Example 16: Finally, the polynomial fitting method was tested on a larger scale graph, with $N = 50$ and $M = 2$. The original and estimated weight matrices are shown in Figure 4.9.

So far, the examples related to classical data analytics have used Fourier analysis and a circular directed graph. The problem formulation presented in this section can also be used to define a graph such that

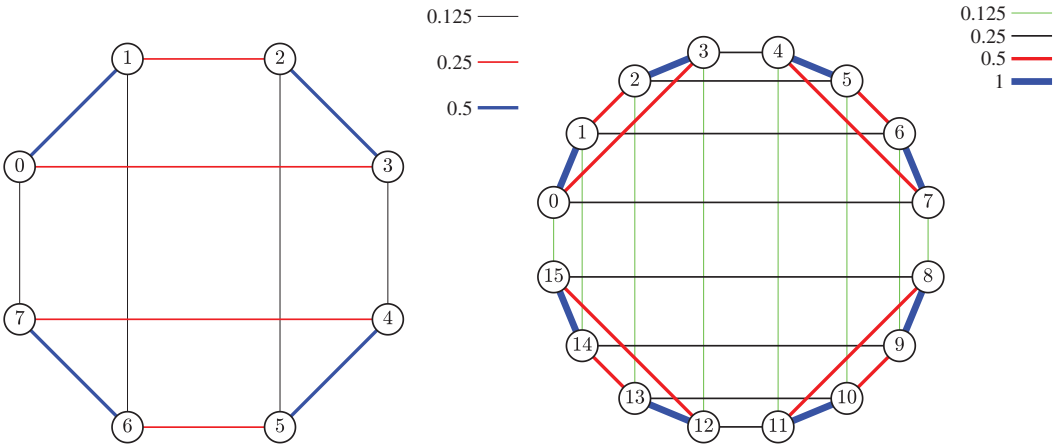


Figure 4.10: Graphs for which the Laplacian eigenvectors are the Hadamard transform basis functions for $N = 8$ (left) and $N = 16$ (right). Different colors and widths of the edges correspond to specific indicated weights.

the spectral analysis on this graph leads to some other well known transforms.

Example 17: We shall illustrate the method of defining a graph which corresponds to the classical Hadamard transform with $N = 8$, and with the eigenvectors

$$\mathbf{U} = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}.$$

If the eigenvalues are found so as to minimize the number of nonzero elements in the Laplacian, we obtain the graphs for $N = 8$ and $N = 16$, as shown in Figure 4.10.

A methodology to estimate the underlying graph topology by means of capturing both spatial and time dependencies among multiple time series was introduced in Mei and Moura (2016), whereby time dependencies among data channels are modeled by an auto-regressive (AR) process, while spatial dependencies are estimated by describing the

matrix coefficients of the AR process as graph polynomial filters. These authors present three algorithms to estimate the graph adjacency matrix and parameters of the graph polynomial filters.

Another important aspect is that the physical process at hand may dictate that graph topologies need to "jump" between a finite number of discrete states, as manifested by sudden changes in their behavior, the problem considered in Baingana and Giannakis (2016). This type of analysis was inspired by the modeling of contagions, such as the spread of popular news stories or infectious diseases, which propagate in cascades over dynamic graphs/networks. For example, an e-mail network may switch topologies from predominantly work-based connections during the week to friend-based connections over the weekend. In such settings, approaches which assume that network dynamics arise as a result of slow topology variations may yield unpredictable results. To this end, Baingana and Giannakis (2016) employ prior knowledge to introduce novel structural equation models with switched dynamics to effectively capture such causal relationships.

5

From Newton Minimization to Graphical LASSO, via LASSO

Most current approaches to the learning of graph topology from the available data are based on the regression method of the least absolute shrinkage and selection operator (LASSO), with its extension to graphs called the graphical LASSO (GLASSO). This class of methods has already been used in the previous section to learn graph topologies. Because of their importance, they will now be derived and explained in detail, starting from simple one-dimensional Newton minimization.

5.1 Newton Method

We shall first briefly review the Newton iterative algorithm for finding the minimum of a convex function. Consider a function, $f(x)$, and assume that it is differentiable. Denote the position of the minimum of $f(x)$ by x^*. The first derivative of $f(x)$ at the minimum point position

$$x^* = x + \Delta x$$

can be expanded into a Taylor series around an arbitrary position x, using the linear model (which is exact if $f'''(x) = 0$ for all x), as

$$f'(x^*) = f'(x) + f''(x)\Delta x. \tag{5.1}$$

Since $f'(x^*) = 0$, by definition, with $\Delta x = x^* - x$, the relation in (5.1) can be rewritten as

$$x^* - x = -\frac{f'(x)}{f''(x)}.$$

This formula is used to define an iterative procedure (called **Newton's iterative method**) for finding the position of the minimum of function $f(x)$, denoted by x^*, starting from an $x = x_0$, as

$$x_{k+1} = x_k - \alpha f'(x_k). \tag{5.2}$$

The parameter α is commonly used instead of $1/f''(x)$ to control the iteration step, and its value should be

$$0 < \alpha \leq \max(|1/f''(x)|),$$

for the considered interval of x. This is the form of the well-known **steepest descent method** for convex function minimization.

Notice that the value $x^* = x - \alpha f'(x)$ would also be obtained as a result of the minimization of a cost function defined by the quadratic form

$$\begin{aligned} x^* &= \arg\min_z G(z) \\ &= \arg\min_z \left(f(x) + f'(x)(z-x) + \frac{1}{2\alpha}(z-x)^2 \right). \end{aligned}$$

Namely, from the zero-value of the derivative of this cost function

$$\frac{d}{dz}\left(f(x) + f'(x)(z-x) + \frac{1}{2\alpha}(z-x)^2 \right) = 0$$

we would arrive at

$$z = x - \alpha f'(x) = x^*.$$

Next, assume that we wish to minimize the cost function

$$J(x) = \frac{1}{2\alpha}(x-y)^2 + \rho|x|,$$

where ρ is a parameter. This cost function corresponds to the minimization of the squared difference between x and y, that is $(x-y)^2$, with

an additional sparsity constraint on x, given by $|x|$. From

$$\frac{dJ(x)}{dx} = \frac{1}{\alpha}(x - y) + \rho \text{sign}(x) = 0$$

we immediately obtain

$$x + \rho\alpha \text{sign}(x) = y.$$

Soft-thresholding, denoted as $\text{soft}(y, \alpha\rho)$, may be used as a solution to this equation, to yield

$$x = \text{soft}(y, \alpha\rho) = \begin{cases} y + \alpha\rho, & \text{for } y < -\alpha\rho \\ 0, & \text{for } |y| \leq \alpha\rho \\ y - \alpha\rho, & \text{for } y > \alpha\rho. \end{cases} \tag{5.3}$$

This form could be considered as the LASSO method for one-dimensional variables. Now, we can proceed with deriving the LASSO method for N-dimensional variables.

5.2 Standard LASSO

For the LASSO minimization with N-dimensional variables, we will consider the cost function

$$\begin{aligned} J(\mathbf{X}) &= \|\mathbf{y} - \mathbf{A}\mathbf{X}\|_2^2 + \rho\|\mathbf{X}\|_1 \\ &= \|\mathbf{y}\|_2^2 - 2\mathbf{X}^T\mathbf{A}^T\mathbf{y} + \mathbf{X}^T\mathbf{A}^T\mathbf{A}\mathbf{X} + \rho\|\mathbf{X}\|_1, \end{aligned}$$

where $\mathbf{y}$ is an $M \times 1$ column vector, $\mathbf{X}$ is an $N \times 1$ column vector, and $\mathbf{A}$ is an $M \times N$ matrix (Stanković, 2015).

The minimization of this cost function with respect to the N-dimensional variable, $\mathbf{X}$, will produce a value which minimizes $\|\mathbf{y} - \mathbf{A}\mathbf{X}\|_2^2$, meaning that $\mathbf{A}\mathbf{X}$ is *as close to* $\mathbf{y}$ as possible, while at the same time *promoting the sparsity* of $\mathbf{X}$, through the term $\|\mathbf{X}\|_1$ in the minimization. The balance between these two requirements is governed by the parameter ρ.

Consider first the differentiable part of the cost function $J(\mathbf{X})$ denoted by

$$J_D(\mathbf{X}) = \|\mathbf{y} - \mathbf{A}\mathbf{X}\|_2^2 = (\mathbf{y} - \mathbf{A}\mathbf{X})^T(\mathbf{y} - \mathbf{A}\mathbf{X}). \tag{5.4}$$

Its derivatives are

$$\frac{\partial J_D(\mathbf{X})}{\partial \mathbf{X}^T} = -2\mathbf{A}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{A}^T\mathbf{A}$$

and

$$\frac{\partial^2 J_D(\mathbf{X})}{(\partial \mathbf{X}^T)^2} = 2\mathbf{A}^T\mathbf{A}.$$

A linear model for the first derivative of $J_D(\mathbf{X})$ around its minimum, $\mathbf{X}^*$, which corresponds to (5.1), is given by

$$\frac{\partial J_D(\mathbf{X}^*)}{\partial \mathbf{X}^T} = \mathbf{0} = \frac{\partial J_D(\mathbf{X})}{\partial \mathbf{X}^T} + (\Delta\mathbf{X})\frac{\partial^2 J_D(\mathbf{X})}{(\partial \mathbf{X}^T)^2}.$$

By replacing the inverse of the second order derivative, $1/(\frac{\partial^2 J_D(\mathbf{X})}{(\partial \mathbf{X}^T)^2})$, by a constant diagonal matrix $\alpha\mathbf{I}$, as in (5.2), we have

$$\Delta\mathbf{X} = \mathbf{X}^* - \mathbf{X} = -\frac{\frac{\partial J_D(\mathbf{X})}{\partial \mathbf{X}^T}}{\frac{\partial^2 J_D(\mathbf{X})}{(\partial \mathbf{X}^T)^2}} = -\alpha\frac{\partial J_D(\mathbf{X})}{\partial \mathbf{X}^T},$$

or

$$\mathbf{X}^* = \mathbf{X} - \alpha\frac{\partial J_D(\mathbf{X})}{\partial \mathbf{X}^T}, \tag{5.5}$$

with

$$0 < \alpha < \frac{1}{\max \|2\mathbf{A}^T\mathbf{A}\|} = \frac{1}{2\lambda_{\max}},$$

where $\lambda_{\max}$ is the maximum eigenvalue of matrix $\mathbf{A}^T\mathbf{A}$.

In order to find $\mathbf{Z} = \mathbf{X}^*$ that minimizes the complete cost function $J(\mathbf{X})$ we can minimize the squared value of the difference

$$\mathbf{Z} - \left(\mathbf{X} - \alpha\mathbf{I}\frac{\partial J_D(\mathbf{X})}{\partial \mathbf{X}^T}\right)$$

and the norm-one of $\mathbf{Z}$, by forming the cost function, $G(\mathbf{Z})$, as

$$G(\mathbf{Z}) = \frac{1}{2\alpha}\left\|\mathbf{Z} - \left(\mathbf{X} - \alpha\mathbf{I}\frac{\partial J_D(\mathbf{X})}{\partial \mathbf{X}^T}\right)\right\|_2^2 + \rho\|\mathbf{Z}\|_1.$$

The minimization of $G(\mathbf{Z})$ will produce $\mathbf{Z}$ which is as close as possible to the desired solution in (5.5), while minimizing its norm-one (maximum sparsity) at the same time, with ρ as the balance parameter.

If we use the notation

$$\mathbf{Y} = \left(\mathbf{X} - \alpha\mathbf{I}\frac{\partial J_D(\mathbf{X})}{\partial \mathbf{X}^T}\right),$$

the solution of

$$\mathbf{X}^* = \arg\min_{\mathbf{Z}} G(\mathbf{Z}) = \arg\min_{\mathbf{Z}} \frac{1}{2\alpha}\|\mathbf{Z} - \mathbf{Y}\|_2^2 + \rho\|\mathbf{Z}\|_1$$

is obtained from

$$\frac{1}{\alpha}(\mathbf{X}^* - \mathbf{Y}) + \rho\text{sign}(\mathbf{X}^*) = \mathbf{0}.$$

Using the soft-thresholding function as in (5.3), we can further write

$$\mathbf{X}^* = \text{soft}(\mathbf{Y}, \alpha\rho).$$

Next, we can replace the value of $\mathbf{Y}$ by

$$\begin{aligned}\mathbf{Y} &= \left(\mathbf{X} - \alpha\mathbf{I}\frac{\partial J_D(\mathbf{X})}{\partial \mathbf{X}^T}\right) = \mathbf{X} - \alpha\mathbf{I}(-2\mathbf{A}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{A}^T\mathbf{A}) \\ &= 2\alpha\mathbf{A}^T\mathbf{y} + (\mathbf{I} - 2\alpha\mathbf{A}^T\mathbf{A})\mathbf{X}.\end{aligned}$$

The iterative formula for the solution of the so defined minimization problem is obtained by replacing $\mathbf{X}^* = \mathbf{X}_{k+1}$ and $\mathbf{X} = \mathbf{X}_k$, to yield

$$\mathbf{X}_{k+1} = \text{soft}(2\alpha\mathbf{A}^T(\mathbf{y} - \mathbf{A}\mathbf{X}_k) + \mathbf{X}_k, \alpha\rho). \tag{5.6}$$

This formula can be rewritten for each element of $\mathbf{X}_k$ and implemented as in Algorithm 1. This is the essence of the LASSO (Least Absolute Shrinkage and Selection Operator) iterative algorithm. Notice that $\mathbf{X}_0 = \mathbf{A}^T\mathbf{y}$ is commonly used as the initial estimate.

Example 18: Consider a sparse signal, $X(k)$, with $N = 60$ elements. In general, to calculate these signal elements we need at least $M = 60$ measurements (linear combinations of signal elements). A signal can be reconstructed from a reduced set of $M < N$ measurements if it is sparse, with $K \ll N$ nonzero elements at unknown positions.

Assume that the original sparse signal of the total length $N = 60$ has the values in the transform domain given by $X(k) = 0$ for all k except for $X(5) = 1$, $X(12) = 0.5$, $X(31) = 0.9$, and $X(45) = -0.75$, and that

it is measured with a matrix $\mathbf{A}$ with only $M = 40 < N$ measurements stored in vector $\mathbf{y}$.

The measurement matrix $\mathbf{A}$ is formed as a Gaussian random matrix of the size 40×60, with elements $\mathcal{N}(0, \sigma^2)$, where $\sigma^2 = 1/40$ is used. All 60 signal values were reconstructed using these 40 measurements, $\mathbf{y}$, and the matrix $\mathbf{A}$, in 1000 iterations. In the initial iteration, $\mathbf{X}_0 = \mathbf{A}^T\mathbf{y}$, was used; then for each next iteration k, the new values of $\mathbf{X}$ were calculated using (5.6) and Algorithm 1, given the data $\mathbf{y}$ and matrix $\mathbf{A}$. The results for $\rho = 0.1$ and $\rho = 0.001$ are shown in Figure 5.1. For a very small $\rho = 0.001$, the result is not sparse, since the constraint is too weak.

5.3 Graphical LASSO

In graph model learning, the corresponding cost function of the form

$$J(\mathbf{Q}) = -\ln(\det \mathbf{Q}) + \text{Trace}(\mathbf{Q}\mathbf{R}_x) + \rho\|\mathbf{Q}\|_1$$

may be used. Here, $\mathbf{Q}$ is the $N \times N$ generalized Laplacian matrix, while $\mathbf{R}_x$ is the available $N \times N$ data correlation matrix. Physical meaning of these terms is explained in Section 4.4.

The derivative of the cost function with respect to the elements of $\mathbf{Q}$ can be written as

$$-\mathbf{Q}^{-1} + \mathbf{R}_x + \rho\text{sign}(\mathbf{Q}) = \mathbf{0} \tag{5.7}$$

at $\partial J(\mathbf{Q})/\partial\mathbf{Q} = \mathbf{0}$.

Upon introducing the notation

$$\mathbf{V} = \mathbf{Q}^{-1}$$

or

$$\mathbf{V}\mathbf{Q} = \mathbf{I}$$

we can write

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_{11} & \mathbf{v}_{12} \\ \mathbf{v}_{12}^T & v_{22} \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{q}_{12} \\ \mathbf{q}_{12}^T & q_{22} \end{bmatrix} \tag{5.8}$$

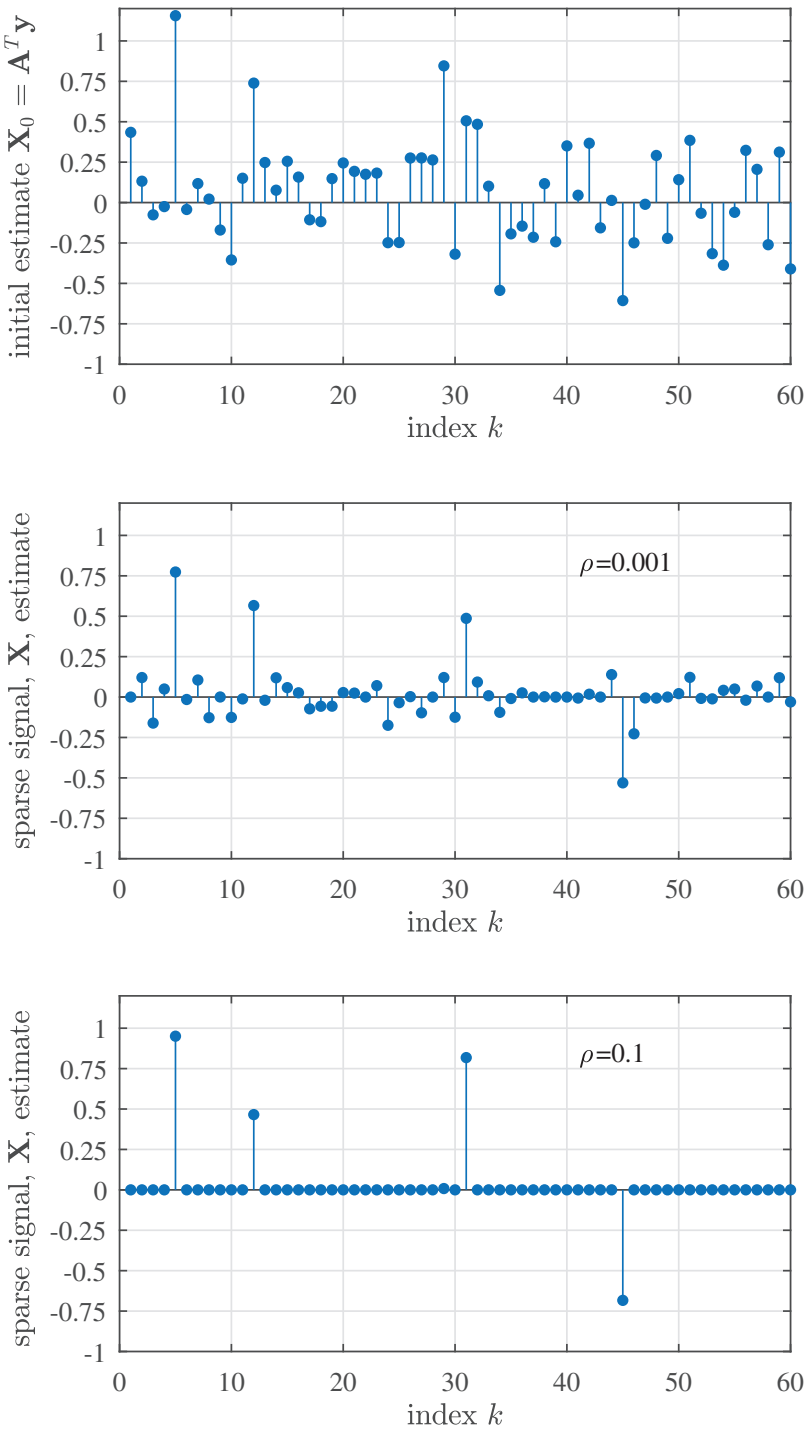


Figure 5.1: A sparse signal with $N = 60$ samples and $K = 4$ nonzero elements, which is reconstructed using a reduced set of $M = 40$ observations and the LASSO iterative algorithm. The top panel shows the result for the matched filter (initial estimate), $\mathbf{X}_0 = \mathbf{A}^T\mathbf{y}$, and the middle and bottom panel for the LASSO iterative algorithm with $\rho = 0.1$ and $\rho = 0.001$. Observe a poor reconstruction for a very small $\rho = 0.001$, which according to (5.6) could not yield a sparse signal.

and

$$\begin{bmatrix} \mathbf{V}_{11} & \mathbf{v}_{12} \\ \mathbf{v}_{12}^T & v_{22} \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{q}_{12} \\ \mathbf{q}_{12}^T & q_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{5.9}$$

where $\mathbf{Q}_{11}$ and $\mathbf{V}_{11}$ are $(N-1) \times (N-1)$ matrices, $\mathbf{v}_{12}$ and $\mathbf{q}_{12}$ are $(N-1) \times 1$ column vectors, and v_{22} and q_{22} are scalars.

After multiplying the first row of blocks in $\mathbf{V}$ with the last column of blocks in $\mathbf{Q}$, we have

$$\mathbf{V}_{11}\mathbf{q}_{12} + \mathbf{v}_{12}q_{22} = \mathbf{0}$$

which gives

$$\mathbf{v}_{12} = -\mathbf{V}_{11}\mathbf{q}_{12}/q_{22} = \mathbf{V}_{11}\boldsymbol{\beta}, \tag{5.10}$$

where

$$\boldsymbol{\beta} = -\mathbf{q}_{12}/q_{22} \tag{5.11}$$

is normalized with $q_{22} > 0$.

Now, from the derivative Equation (5.7) we may write

$$-\begin{bmatrix} \mathbf{V}_{11} & \mathbf{v}_{12} \\ \mathbf{v}_{12}^T & v_{22} \end{bmatrix} + \begin{bmatrix} \mathbf{R}_{11} & \mathbf{r}_{12} \\ \mathbf{r}_{12}^T & r_{22} \end{bmatrix} + \rho \text{sign}\left(\begin{bmatrix} \mathbf{Q}_{11} & \mathbf{q}_{12} \\ \mathbf{q}_{12}^T & q_{22} \end{bmatrix} \right) = \mathbf{0}.$$

For the upper right block we have

$$-\mathbf{v}_{12} + \mathbf{r}_{12} + \rho \text{sign}(\mathbf{q}_{12}) = \mathbf{0},$$

while after replacing $\mathbf{v}_{12} = \mathbf{V}_{11}\boldsymbol{\beta}$ and $\mathbf{q}_{12} = -\boldsymbol{\beta}/q_{22}$ from (5.10) and (5.11) we arrive at

$$-\mathbf{V}_{11}\boldsymbol{\beta} + \mathbf{r}_{12} - \rho \text{sign}(\boldsymbol{\beta}) = \mathbf{0}. \tag{5.12}$$

The solution to this equation for $\boldsymbol{\beta}$ has been already defined within the LASSO framework, and is given by

$$\beta_i V_{11}(i) = \text{soft}\left(r_{12}(i) - \sum_{k \neq i} V_{11}(k,i)\beta_k, \rho \right). \tag{5.13}$$

In order to apply the LASSO as in (5.6), we can interpret the minimization of the difference

$$\mathbf{A}^T(\mathbf{y} - \mathbf{A}\mathbf{X}) = \mathbf{A}^T\mathbf{y} - \mathbf{A}^T\mathbf{A}\mathbf{X}$$

in (5.6) as the task of finding the least-squares estimate of $\mathbf{A}^T\mathbf{y}$ by $\mathbf{A}^T\mathbf{A}\mathbf{X}$. Now, we can adjust (5.12) to assume a similar form

$$-\mathbf{V}_{11}^{1/2}\mathbf{V}_{11}^{1/2}\boldsymbol{\beta} + \mathbf{V}_{11}^{1/2}\mathbf{V}_{11}^{-1/2}\mathbf{r}_{12} - \rho\text{sign}(\boldsymbol{\beta}) = \mathbf{0}. \tag{5.14}$$

In this case, the matrix $\mathbf{V}_{11}^{1/2}$ plays the role of $\mathbf{A}$ in (5.6) and $\mathbf{V}_{11}^{-1/2}\mathbf{r}_{12}$ plays the role of $\mathbf{y}$. Therefore, the standard LASSO should be calculated using

$$\boldsymbol{\beta} = \text{lasso}(\mathbf{V}_{11}^{1/2}, \mathbf{V}_{11}^{-1/2}\mathbf{r}_{12}, \rho) \tag{5.15}$$

as in Algorithm 3.

Now, the graphical LASSO (GLASSO) iterative algorithm can be summarized as follows.

- In the initial step, use

$$\mathbf{V} = \mathbf{R}_x + \rho\mathbf{I}.$$

- For each coordinate, $j = 1, 2, \ldots, N$, the matrix equation of the form (5.9) is written. For each j, the reduced matrix $\mathbf{V}_{11}$ is formed by omitting the jth row and the jth column. Then, the matrix $\mathbf{R}_x$ is rearranged accordingly.
- Equation (5.13) is solved using (5.15).
- The matrix $\mathbf{V}$ is updated for each j by inserting the jth column

$$\mathbf{v}_{12} = \mathbf{V}_{11}\boldsymbol{\beta},$$

  and inserting at the jth row $\mathbf{v}_{12}^T$ with the element v_{22} at the jth position.
- After all j indices are used in the calculation, the final estimate of the generalized Laplacian is obtained as $\mathbf{Q} = \mathbf{V}^{-1}$.

This calculation procedure is also presented in Algorithm 3.

Remark 11: Notice that the value of matrix $\mathbf{Q} = \mathbf{V}^{-1}$ is updated for each j and in the last iteration, using the column vector

$$\mathbf{q}_{12} = -\boldsymbol{\beta}q_{22},$$

where q_{22} can be calculated from $\mathbf{v}_{12}^T \mathbf{q}_{12} + v_{22} q_{22} = 1$ or $-\mathbf{v}_{12}^T \boldsymbol{\beta} q_{22} + v_{22} q_{22} = 1$, finally producing the values

$$q_{22} = \frac{1}{v_{22} - \mathbf{v}_{12}^T \boldsymbol{\beta}},$$

and

$$\mathbf{q}_{12} = \frac{\boldsymbol{\beta}}{\mathbf{v}_{12}^T \boldsymbol{\beta} - v_{22}}$$

which are used to update the jth column and row of the matrix $\mathbf{Q}$ in the same as the update of matrix $\mathbf{V}$.

This algorithm can be used for iterative matrix inversion with $\rho = 0$.

Example 19: Consider a graph with $N = 50$ vertices and with a small number of edges, so that the weight matrix, $\mathbf{W}$, is sparse. The ground truth weight matrix, $\mathbf{W}$, is shown in Figure 5.2(a). This matrix was then estimated from a large number, $P = 1000$, of observations of a signal on this graph. Both the precision matrix, $\mathbf{R}^{-1}$, and the graphical LASSO, given in Figures 5.2(b) and (c), produce good estimation of the weight matrix, $\mathbf{W}$. Next, the number of observations was significantly reduced to $P = 40 < N = 50$, a case when the correlation matrix, $\mathbf{R}$, is singular and of rank lower or equal to $P = 40$. In this case, the sparsity of the weight matrix is crucial for the solution. Here, only the graphical LASSO, which includes the sparsity constraint, was able to produce good result, as shown in Figure 5.2(e), while the precision matrix could be calculated only through a pseudo-inverse, and cannot be used as the weight matrix estimate, as can be seen from Figure 5.2(d).

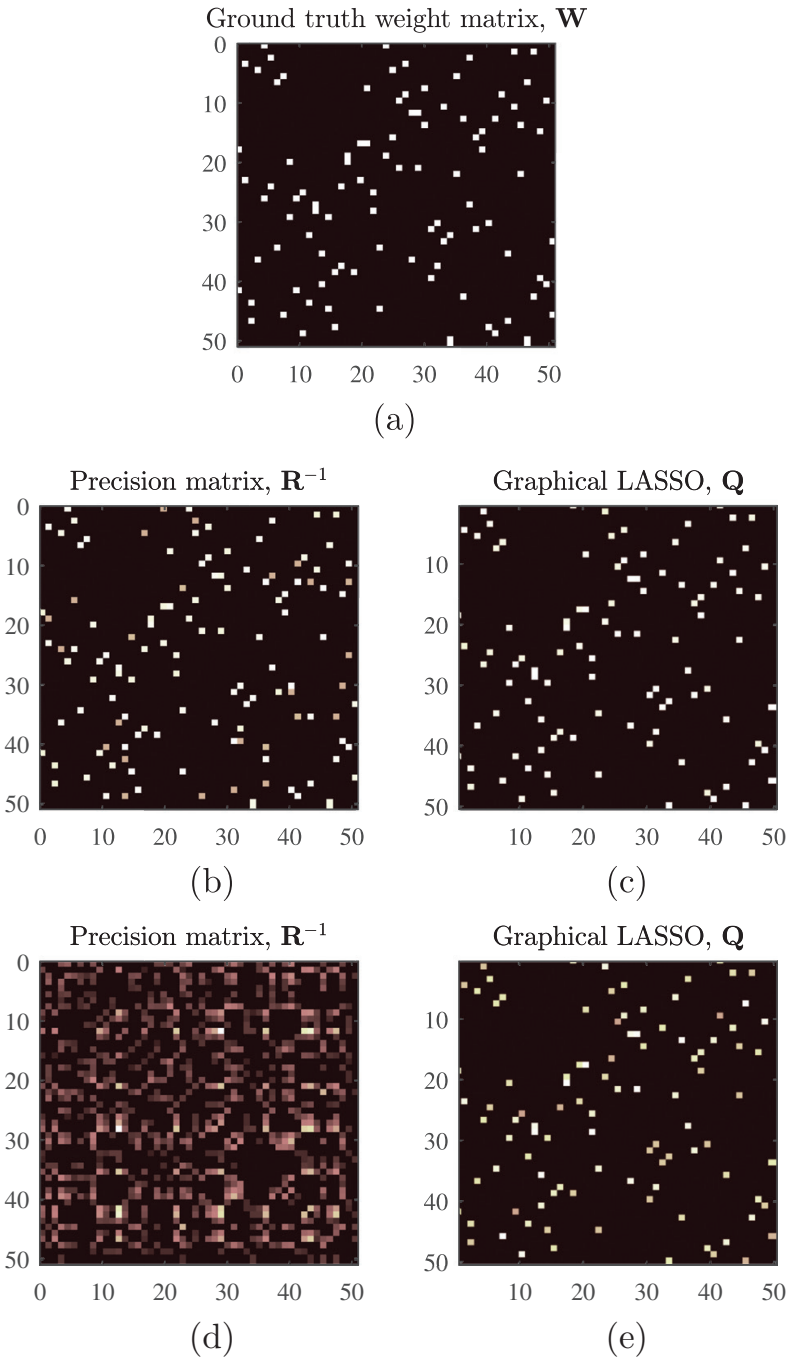


Figure 5.2: Estimation of the weight matrix, $\mathbf{W}$, for a graph with $N = 50$ randomly positioned vertices. (a) Ground truth weight matrix, $\mathbf{W}$. (b) Precision matrix, for a large number of observations, $P = 1000 \gg N = 50$. (c) Estimated weight matrix using the graphical LASSO, for a large number of observations, $P = 1000 \gg N = 50$. (d) Precision matrix, for a small number of observations, $P = 40 < N = 50$ (the correlation matrix, $\mathbf{R}$, is singular and with a rank lower or equal to P, so that pseudo-inversion is used). (e) Estimated weight matrix using the graphical LASSO, for a small number of observations, $P = 40 < N = 50$.

6

Physically Well Defined Graphs

The simplest scenario for graph connectivity consideration is when the graph associated with a problem at hand is *physically well defined*. Examples of such graphs are manifold, including electric circuits, power networks, linear heat transfer, social and computer networks, and spring-mass systems, all of which will be addressed in this section.

6.1 Resistive Electrical Circuits

Graph theory based methods for the analysis and transformations of electrical circuits have long been part of classical courses and textbooks. It is also interesting that some general information theory problems can be interpreted and solved within the graph approach to basic electric circuits. In such cases, the underlying graph topology is well defined and is a part of the problem statement.

The graph Laplacian can also be considered within the basic electric circuit theory. In this case, since the graph Laplacian can be derived based on the Kirchhoff's laws, it is also known as the *Kirchhoff matrix*.

Graph Representation of Electric Circuits

Consider a resistive electric circuit, and the electric potential in the circuit vertices (nodes), denoted by $x(n)$. The vertices in an electrical

circuit are connected with edges, where the weight of an edge connecting the vertices n and m is defined by the edge conductance, W_{nm}. The conductances are the reciprocal values to edge resistances

$$W_{nm} = \frac{1}{R_{nm}}.$$

The current in the edge from vertex n to vertex m is then equal to

$$i_{nm} = \frac{x(n) - x(m)}{R_{nm}} = W_{nm}(x(n) - x(m)).$$

In addition to the edge currents, an external current generator may be attached to every vertex, and can be considered as a source of signal change in the vertices; the external current at a vertex n is denoted by i_n.

Since the sum of all currents going in/from a vertex n, $n = 0, 1, \ldots, N-1$, must be 0, that is

$$-i_n + \sum_m i_{nm} = 0,$$

the current of the external generator at a vertex n must be equal to the sum of all edge currents going in/from this vertex, to give

$$i_n = \sum_m W_{nm}(x(n) - x(m)) = d_n x(n) - \sum_m W_{nm} x(m), \qquad n = 0, 1, \ldots, N-1,$$

where

$$d_n = \sum_m W_{nm} = \sum_{m=0}^{N-1} W_{nm}$$

is the degree of vertex n. The summation over m can be extended to all vertices, $m = 0, 1, \ldots, N-1$, since $W_{nm} = 0$ if there is no edge between vertices n and m.

The above equations can be written in a matrix form as

$$\mathbf{i} = \mathbf{D}\mathbf{x} - \mathbf{W}\mathbf{x}$$

or

$$\mathbf{L}\mathbf{x} = \mathbf{i} \tag{6.1}$$

where $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the Laplacian of a graph representing an electric circuit and $\mathbf{i}$ is the vector of currents at every vertex.

If the Laplacian matrix is decomposed as $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, from (6.1) we have $\mathbf{\Lambda}\mathbf{U}^T\mathbf{x} = \mathbf{U}^T\mathbf{i}$, and

$$\mathbf{\Lambda}\mathbf{X} = \mathbf{I}, \tag{6.2}$$

where $\mathbf{X} = \mathbf{U}^T\mathbf{x}$ and $\mathbf{I} = \mathbf{U}^T\mathbf{i}$ are the GDFT of graph signals $\mathbf{x}$ and $\mathbf{i}$ (see Part II, Section 3.6).

From (6.2), the components of the spectral transform vector, $\mathbf{X}$, satisfy

$$\lambda_k X(k) = I(k)$$

for each k.

A signal measured on an electrical circuit graph can be related to the above theory in several ways. For example, potentials on all vertices could be measured under some measurement noise, which calls for data filtering on a graph. Another possible case is when external conditions are imposed, for example external sources are applied to some vertices. We are then interested in the values of electric potential at all vertices, a problem which corresponds to graph signal reconstruction.

For nontrivial solutions, there should be an external source on at least two vertices. If we assume that a vertex with an external source is chosen as a reference vertex, then the signal or external source values at these vertices (with external sources) are sufficient to find signal values at all other vertices.

Example 20: Consider the graph and signal sensed on the graph presented in Figure 6.1. The signal values are

$$\mathbf{x} = [6.71, 6.88, 7.13, 5.25, 6.67, 8.18, 2.62, 0]^T$$

and the graph Laplacian (as a matrix operator) applied to the signal yields

$$\mathbf{L}\mathbf{x} = [0, 0, 1, 0, 0, 2, 0, -3]^T.$$

Observe that in this case the vertices indexed by $0, 1, 3, 4, 6$ are *not active*, and their values can be obtained as linear combinations of the

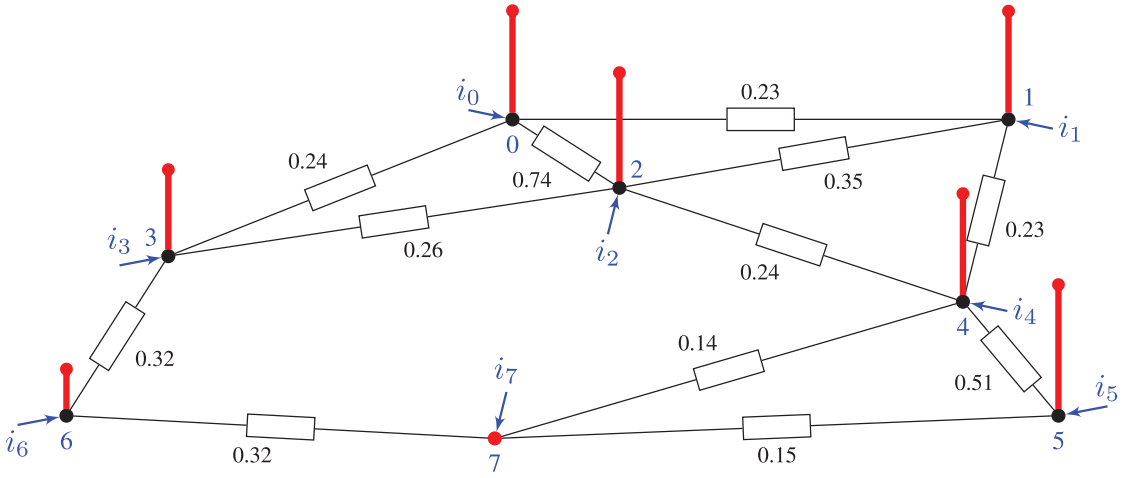


Figure 6.1: Electric potential, $x(n)$, as a signal on an electric circuit graph.

signals at neighboring active vertices, that is

$$\begin{aligned} 1.21x(0) - 0.23x(1) - 0.74x(2) - 0.24x(3) &= 0 \\ -0.23x(0) + 0.81x(1) - 0.35x(2) - 0.23x(4) &= 0 \\ -0.24x(0) - 0.26x(2) + 0.82x(3) - 0.32x(6) &= 0 \\ -0.23x(1) - 0.24x(2) + 1.12x(4) - 0.51x(5) - 0.14x(7) &= 0 \\ -0.32x(3) + 0.64x(6) - 0.32x(7) &= 0. \end{aligned} \tag{6.3}$$

After solving this system with known signal values $x(2) = 7.13$, $x(5) = 8.18$, and $x(7) = 0$ at the active vertices, we obtain the remaining signal values

$$\mathbf{x}_p = [x(0),\ x(1),\ x(3),\ x(4),\ x(6)]^T = [6.71,\ 6.88,\ 5.25,\ 6.67,\ 2.62]^T.$$

Graph Transformations

A graph with one or more inactive vertices (where the elements of $\mathbf{Lx}$ are equal to zero) can be simplified by removing these vertices using the well-known transformations of edges connected in series, parallel, or star-to-mesh transforms. This process corresponds to the *downsampling* of the graph signal (see also Part II). The reduction of an electrical network via a Schur complement of the associated conductance matrix is known as the Kron reduction, whereby the vertices are separated into two groups: active vertices and inner vertices. The inner vertices can

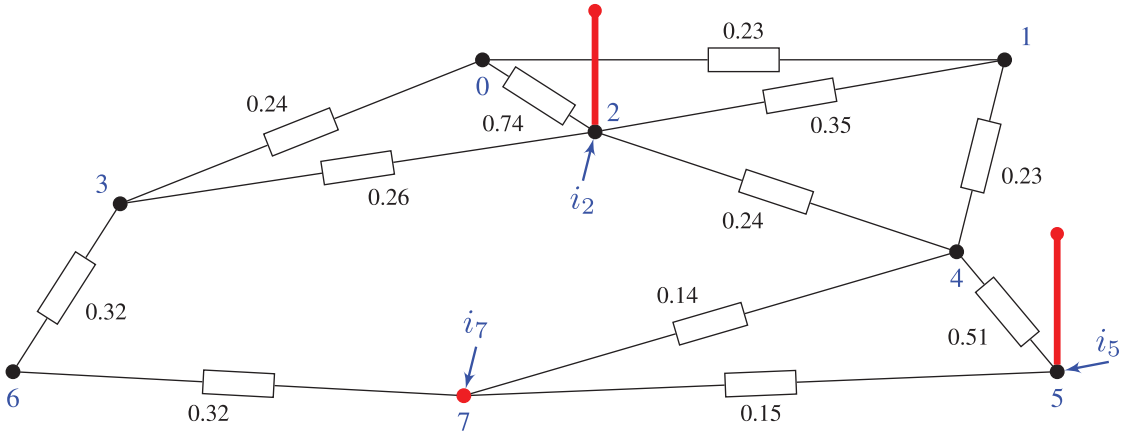


Figure 6.2: Electric potential, $x(n)$, as a signal on an electric circuit graph observed at the three vertices with nonzero external sources. For this graph, all other values of $x(n)$ in Figure 6.1 can be calculated based on the signal values at vertices $n = 2$, $n = 5$, and $n = 7$.

be eliminated from the graph without changing the electric network conditions; this is achieved via equivalent transformations, such as the "star-mesh" transformations (Dorfler and Bullo, 2012).

Similar procedure can be used to add inactive vertices, either by inserting a vertex within an edge or by transforming meshes to stars, in what corresponds to the *interpolation* of the graph signal.

Example 21: For the graph and the graph signal from Example 20, the active vertices are $n = 2, 5, 7$, as shown in Figure 6.2, while the signal values at all vertices are given in Figure 6.1. Notice that the existing signal values will not change, for the given external sources, if the graph is "*downsampled*", as shown in Figure 6.3, or if the graph signal is "*interpolated*" by adding new vertices, as shown in Figure 6.4. This is closely related to link prediction, that will be addressed later.

Graph Data Denoising for Sparse External Sources

The set of external sources are considered sparse if their number is much smaller than the number of vertices, N. For this scenario, the norm-zero of the external sources vector, $\mathbf{Lx}$, is such that $\|\mathbf{Lx}\|_0 \ll N$. If the noisy observations, $\mathbf{y}$, of data on graph, $\mathbf{x}$, are available and we know that the number of external sources is small, then the cost function for denoising

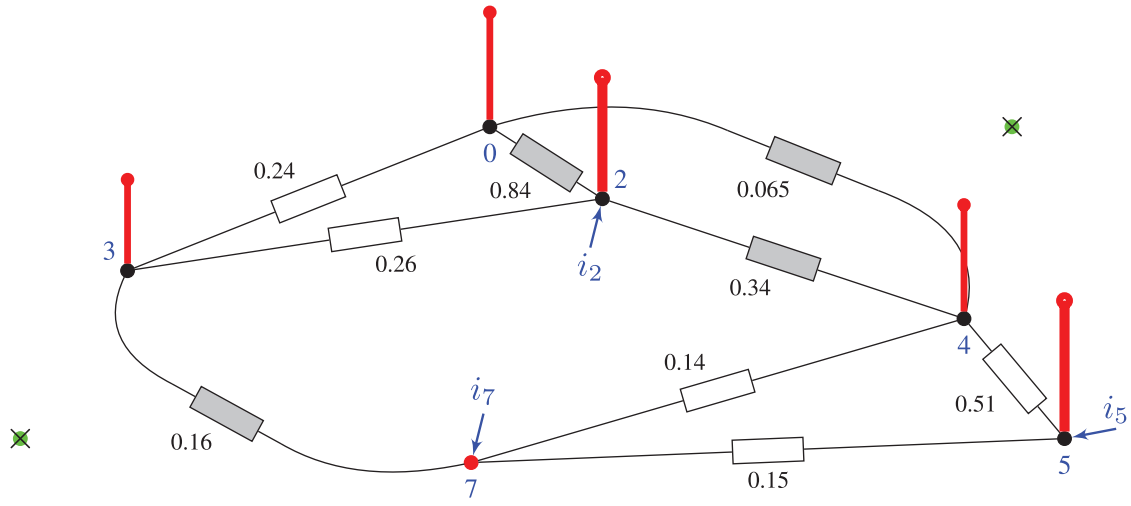


Figure 6.3: Graph signal, $x(n)$, from Figure 6.1, observed on a graph with a reduced number of vertices ("downsampling"), whereby the vertices $n = 6$ and $n = 1$ are removed (*crosses in green dots*). Observe that the signal values at the active vertices, $n = 2$, $n = 5$, and $n = 7$, are not changed. The edge weights in gray shade are the equivalent values obtained using the standard resistor, $R_{mn} = 1/W_{mn}$, transformations.

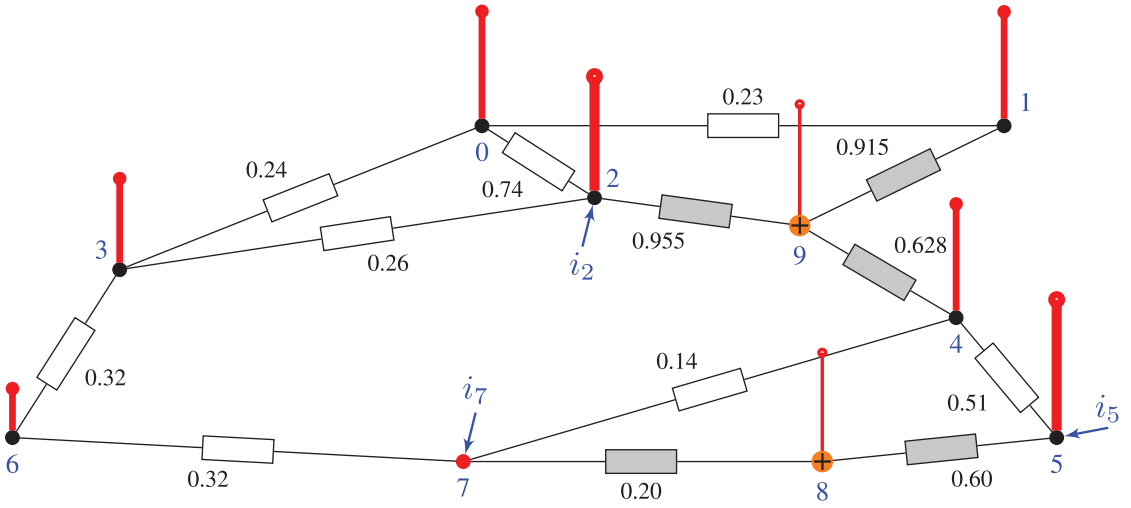


Figure 6.4: Graph signal, $x(n)$, from Figure 6.1 observed on a graph with an extended number of vertices ("interpolation"). Observe that the signal values at all vertices, $n = 0, 1, 2, 3, 4, 5, 6, 7$, from Figure 6.2 are not changed. In the locations where the new vertices $n = 8$ and $n = 9$ are added, the graph signal is interpolated using $x(2)$, $x(5)$, and $x(7)$, as in (6.3), and the corresponding edge weights are shown in gray.

can be written in the form

$$J = \|\mathbf{y} - \mathbf{x}\|_2^2 + \rho\|\mathbf{L}\mathbf{x}\|_0. \tag{6.4}$$

This minimization problem can be solved either using the corresponding norm-one form

$$J = \|\mathbf{y} - \mathbf{x}\|_2^2 + \rho\|\mathbf{L}\mathbf{x}\|_1 \tag{6.5}$$

or using a kind of matching pursuit, as presented in the next example within a classical data denoising scenario.

Example 22: Consider the classical time domain and a piece-wise linear signal, of which noisy observations are available, as shown in Figure 6.5(a). In standard analysis, the graph representation of the domain of this signal is an undirected and unweighted path graph, where the elements of $\mathbf{Lx}$ play the role of external sources, as shown in Figure 6.5(b). We shall assume that $n = 0$ is the reference vertex with $x(0) = 0$.

The data denoising problem is then solved in the following way. The initial estimate of the external sources is calculated as $\mathbf{Ly}$. Since we assumed that the external sources are sparse, we will consider the positions, k_1, k_2, k_3, k_4, and k_5, of $K = 5$ largest absolute values of the initial estimate.

The largest K nonzero values of the external source vector, $\mathbf{Ly}$, are denoted by $\mathbf{J}_K$, with the elements $i(k_1), i(k_2), \ldots, i(k_K)$. The value of $\mathbf{J}_K$ is found in such a way that it minimizes the difference between the estimated data, $\mathbf{L}_K^{(-1)}\mathbf{J}_K$, and the observations, $\mathbf{y}$, that is

$$\min_{\mathbf{J}_K} \|\mathbf{y} - \mathbf{L}_K^{(-1)}\mathbf{J}_K\|_2^2,$$

where $\mathbf{L}_K^{(-1)}$ is obtained from the inverse transform of the graph Laplacian (after the reference row and column, at $n = 0$ are omitted) by keeping only K columns which correspond to the nonzero positions in the external source vector, $\mathbf{J}_K$. The solution therefore becomes

$$\mathbf{J}_K = \text{pinv}(\mathbf{L}_K^{(-1)})\mathbf{y}.$$

After the nonzero external sources are found, the full external source vector, $[i(1), i(2), \ldots, i(N-1)]^T$, is formed using the calculated nonzero values in $\mathbf{J}_K$ and inserting zero values at the remaining positions, as shown in Figure 6.5(c).

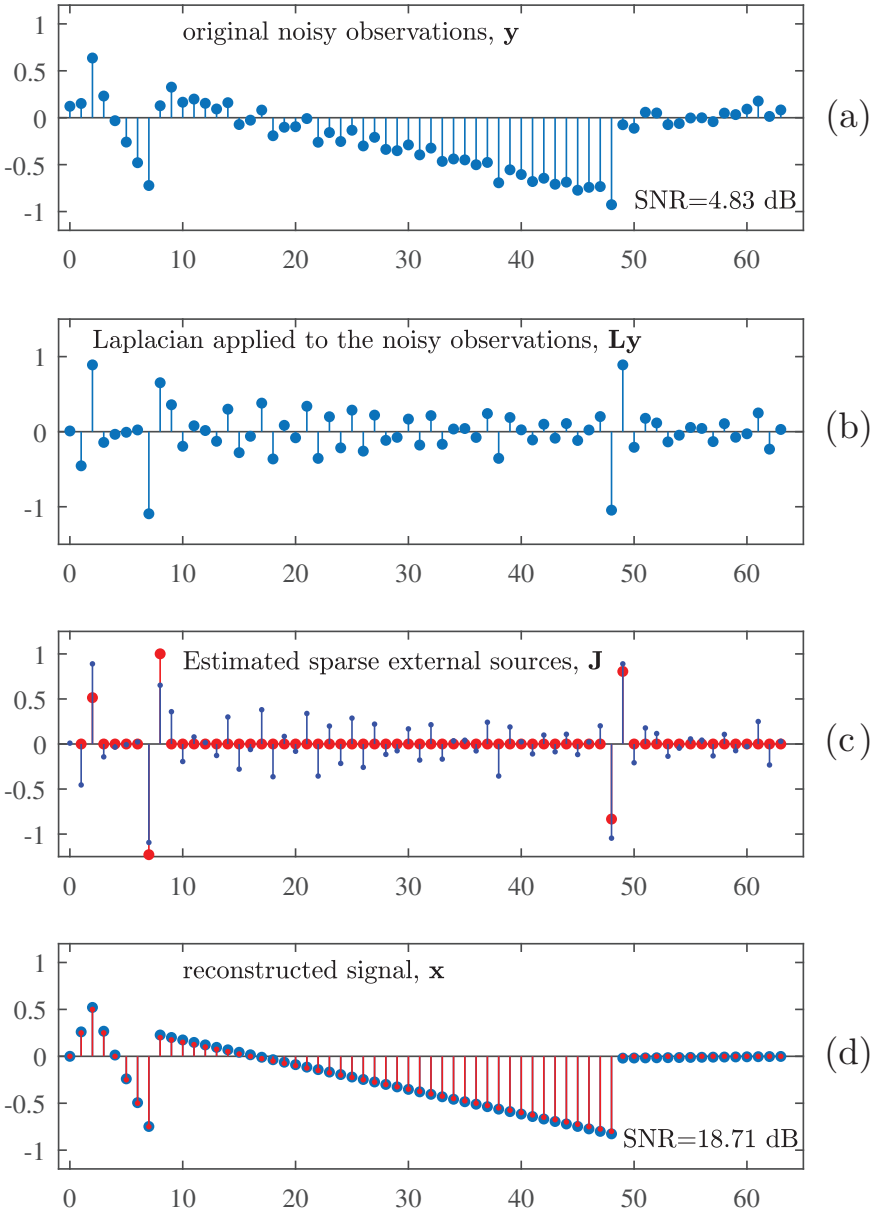


Figure 6.5: Original piece-wise linear noisy signal (top) and the reconstructed signal (bottom), with the Laplacian of the noisy observations and its re-estimated sparse version (middle panels).

Finally, the reconstructed signal is obtained from

$$\begin{bmatrix} L_{11} & L_{12} & \dots & L_{1,N-1} \\ L_{21} & L_{22} & \dots & L_{2,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ L_{N-1,2} & L_{N-1,2} & \dots & L_{N-1,N-1} \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix} = \begin{bmatrix} i(1) \\ i(2) \\ \vdots \\ i(N-1) \end{bmatrix}$$

as, $\mathbf{x} = \mathbf{L}^{-1}\mathbf{J}$, with the result shown in Figure 6.5(d). Note that the matrix $\mathbf{L}$ is obtained from the graph Laplacian matrix by removing the reference vertex row and column.

Remark 12: The crucial advantage over the standard total variation (TV) minimization approach in the compressive sensing based denoising is that the cost function used in Example 22 does not penalize for the linear changes of the signal, while the TV approach promotes piece-wise constant signals.

6.2 Heat Transfer

The same model as in resistive electrical circuits can be used for a heat transfer network. In this case, the signal values are the measured temperatures, $x(n) = T(n)$, while the heat flux is defined as

$$q_{nm} = (T(n) - T(m))C_{nm} = (x(n) - x(m))W_{nm},$$

where C_{nm} are the heat transfer constants, which represent edge weights in the underlying graph, $C_{nm} = W_{nm}$.

Then, the input heat flux in a vertex n can be written as

$$q_n = \sum_m W_{nm}(x(n) - x(m)) = d_n x(n) - \sum_{m=0}^{N-1} W_{nm}x(m),$$

with

$$\mathbf{q} = \mathbf{L}\mathbf{x}.$$

Active vertices are those with an external heat flux, while the passive vertices are those where all heat flux coming to a vertex is forwarded to other vertices, through the edges. An example of a heat transfer graph is given in Figure 6.6.

6.3 Spring-Mass Systems

A spring mass system can also be modeled as a graph. Consider a system of $N = 4$ masses which correspond to a path graph, as in Figure 6.7. Assume that all displacements and forces are aligned with springs. According to Hook's law, the displacements, $x(n)$, and the

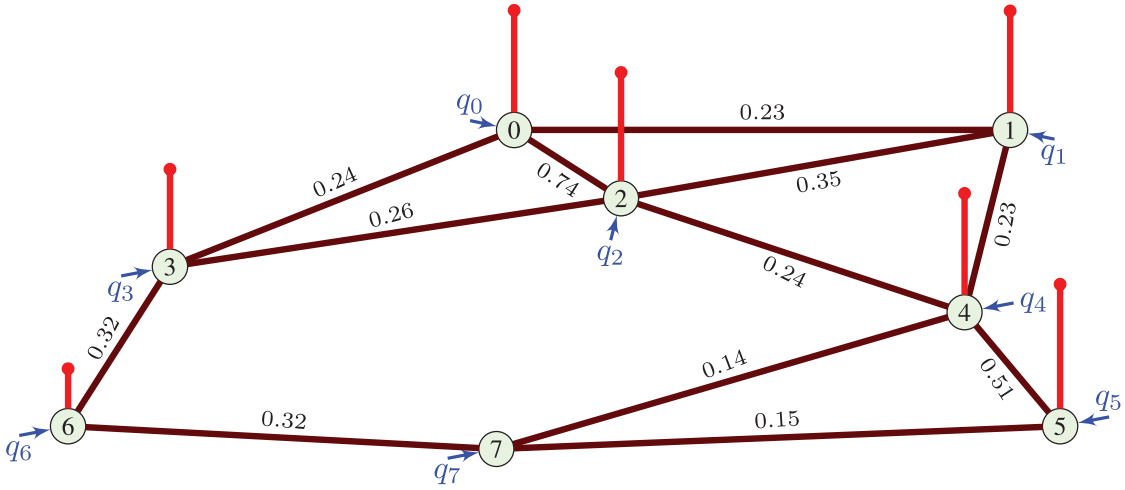


Figure 6.6: Temperature, $x(n) = T(n)$, as a signal on a heat transfer graph, with q_1, q_2, ..., q_7, as the external heat flux values at the corresponding vertices.

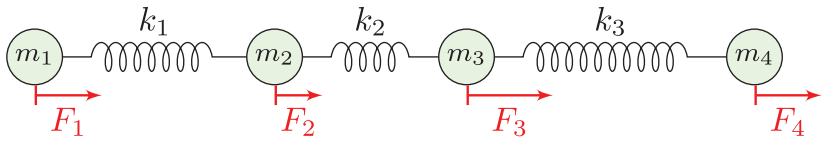


Figure 6.7: Spring-mass system on a path graph.

forces, F_n, at the steady state are related as

$$\begin{aligned} k_1(x(1) - x(2)) &= F_1 \\ k_1(x(2) - x(1)) + k_2(x(2) - x(3)) &= F_2 \\ k_2(x(3) - x(2)) + k_3(x(3) - x(4)) &= F_3 \\ k_3(x(4) - x(3)) &= F_4 \end{aligned}$$

or in a matrix form

$$\begin{bmatrix} k_1 & -k_1 & 0 & 0 \\ -k_1 & k_1 + k_2 & -k_2 & \\ 0 & -k_2 & k_2 + k_3 & -k_3 \\ 0 & 0 & -k_3 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix}$$

$$\mathbf{L}\mathbf{x} = \mathbf{F}.$$

These equations define a weighted graph and its corresponding graph Laplacian.

Given that the graph Laplacian is a singular matrix, in order to solve this system for unknown displacements (graph signal), we should

introduce a reference vertex with a fixed position (zero displacement). Then, the system $\mathbf{Lx} = \mathbf{F}$ can be solved.

6.4 Social Networks and Linked Pages

Social networks are also examples of well defined graphs, where the vertices are network members and the edges define their relationships within a social network. If two members in a social network are related, then the corresponding edge weight is 1, and the weight matrix is equal to the adjacency matrix. An example of a small social network with the corresponding member links is shown in Figure 6.8.

Pages with hyper-links can also be considered as a well defined directed graph; an example of links between $N = 8$ pages is given in Figure 6.9. An interesting parameter for this kind of graphs is the PageRank.

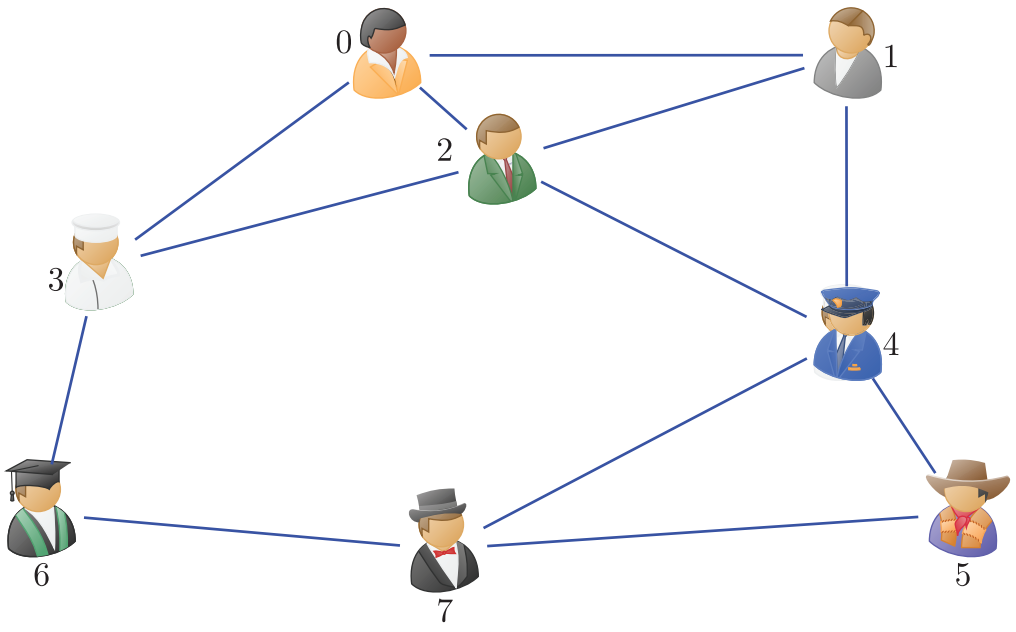


Figure 6.8: An example of a small social network represented as an undirected graph.

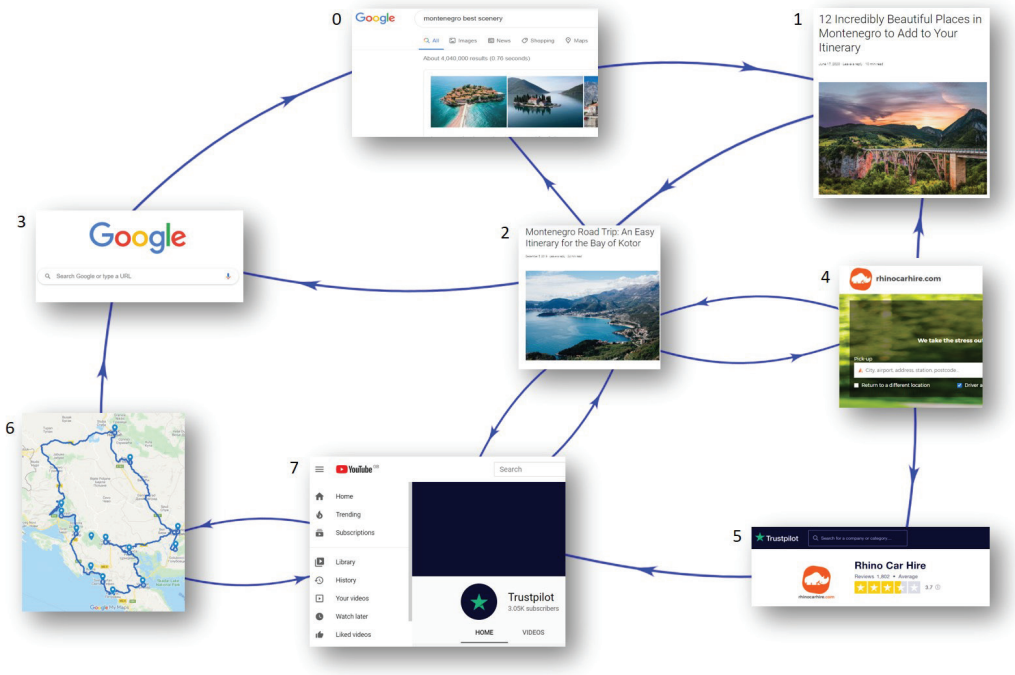


Figure 6.9: Hyper-linked internet pages in a holiday search scenario, which can be represented as a directed graph. The vertices 0 to 7 reflect the nature of web search, as follows. 0: Venue search results. 1: Places to visit at the venue from 0. 2: Search for road trip. 3: Google search engine. 4: Car hire website. 5: Trustpilot ranking of car rental agent. 6: Personalized roadmap for the trip. 7: Review of sites of interest.

6.5 PageRank

The PageRank was defined by Google to rank the web pages. For a directed graph, PageRank of a vertex n is defined as a graph signal satisfying the relation

$$x(n) = \sum_m \frac{1}{d_m} W_{mn} x(m),$$

where W_{mn} are weights of the directed edges connecting vertices m and n, and d_m is the outgoing degree of a vertex m. This means that the PageRank of each vertex is related to the PageRank of the vertices connected to it.

The PageRank is usually calculated using an iterative procedure defined by

$$x_{k+1}(n) = \sum_m \frac{1}{d_m} W_{mn} x_k(m), \tag{6.6}$$

starting from an arbitrary PageRank, for example $x_0(n) = 1$. In the original definition by Google, the scaling factors of 0.15 and 0.85 were added, to give

$$x_{k+1}(n) = 0.15 + 0.85 \sum_m \frac{1}{d_m} W_{mn} x_k(m). \tag{6.7}$$

Example 23: Consider the graph from Figure 6.9 (the same graph as in Part I, Figure 2.1(b)). In this case, the vertices represent pages on the Internet, while the directed edges designate their relations. For example, the page which corresponds to vertex 0 cites (gives a hyper-link to) page marked with 1, while it is cited (hyper-linked) by pages at vertex 2 and vertex 3. All other vertices are connected by the edges in the same way. Intuitively, we can expect that the rank in this network is higher for the pages that are highly cited (hyper-linked) with other also highly cited (hyper-linked) pages. To find the rank of the pages in this graph/network, we first need to calculate the PageRank for all pages/vertices. The weight/adjacency matrix of this graph, $\mathbf{W} = \mathbf{A}$, is given by (see also Part I, Equation (2.2))

$$\mathbf{W} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}. \tag{6.8}$$

The outgoing vertex degrees are calculated as sums of columns of the matrix $\mathbf{W}^T$, that is $d_m = \sum_{n=0}^{7} W_{mn}$, with their values

$$\mathbf{d} = [1 \;\; 1 \;\; 4 \;\; 1 \;\; 3 \;\; 1 \;\; 2 \;\; 2].$$

Now, the PageRank values for vertices can be obtained through an iterative procedure, as in (6.6), starting with the initial page ranks $\mathbf{x}_0 = [1, 1, 1, 1, 1, 1, 1, 1]$. After a few iterations, the results for PageRank

are as follows

$$\begin{bmatrix} \mathbf{x}_0^T \\ \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_5^T \\ \vdots \\ \mathbf{x}_{11}^T \end{bmatrix} = \begin{bmatrix} 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 \\ 1.25 & 1.33 & 1.83 & 0.75 & 0.25 & 0.33 & 0.50 & 1.75 \\ 1.21 & 1.33 & 2.29 & 0.71 & 0.46 & 0.08 & 0.87 & 1.04 \\ & & & & \vdots & & & \\ 1.29 & 1.68 & 2.10 & 0.80 & 0.52 & 0.17 & 0.46 & 0.99 \\ & & & & \vdots & & & \\ 1.33 & 1.53 & 2.14 & 0.80 & 0.55 & 0.18 & 0.48 & 0.99 \end{bmatrix}.$$

The matrix form of the iterations in (6.6) is

$$\mathbf{x}_{k+1} = \mathbf{W}_N \mathbf{x}_k,$$

where $\mathbf{W}_N$ is obtained from $\mathbf{W}^T$ by dividing all elements of the mth column, $m = 0, 1, \ldots, N-1$, by d_m. The mean-values of matrix $\mathbf{W}_N$ columns are normalized.

Example 24: The normalized adjacency/weighting matrix from Example 23, is

$$\mathbf{W}_N = \begin{bmatrix} 0 & 0 & \frac{1}{4} & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{4} & 0 & 0 & 1 & \frac{1}{2} & 0 \end{bmatrix}.$$

The final, steady state, PageRank $\mathbf{x}$ can then be obtained from

$$\mathbf{x} = \mathbf{W}_N \mathbf{x},$$

and represents the eigenvector of the matrix $\mathbf{W}_N$ which corresponds to the eigenvalue equal to 1.

Example 25: The eigenvalue decomposition of the matrix $\mathbf{W}_N$ in Example 23 results in the eigenvector which corresponds to eigenvalue $\lambda_k = 1$, whose elements are

$$\mathbf{x}^T = [1.33 \quad 1.52 \quad 2.18 \quad 0.79 \quad 0.55 \quad 0.18 \quad 0.48 \quad 0.97].$$

The eigenvector is normalized by its mean value, and is obtained via the iterative solution after 11 iterations.

6.6 Random Walk

Assume that the signal, $x(n)$, represents the probability that a random walker is present at a vertex n. The random walker will then transit from the vertex n to one of its neighboring vertices, m, with probability p_{nm}. There are several ways to define this probability and the corresponding forms of random walk; for an extensive review see Masuda *et al.* (2017). Here, we consider two random-walk definitions:

- vertex-centric random walk, and
- edge-centric random walk.

In the **vertex-centric random walk**, the probability, p_{nm}, that a random walker will transit from the vertex n to one of its neighboring vertices, m, is defined by

$$p_{nm} = \frac{W_{nm}}{\sum_m W_{nm}} = \frac{1}{d_n} W_{nm}, \tag{6.9}$$

where W_{nm} are the affinities of the walker to transit from a vertex n to a vertex m, and $d_n = \sum_m W_{nm}$ is the degree of a vertex n. The probability, $x_{p+1}(m)$, that a walker is at the vertex m at the time step $(p+1)$ is then equal to the sum of all probabilities that a walker was at one the vertices, n, with the distance equal to one (neighboring vertices to the vertex m) multiplied by the probabilities that the walker transits from the vertex n to the vertex m, that is

$$x_{p+1}(m) = \sum_n x_p(n) p_{nm} = \sum_n x_p(n) \frac{1}{d_n} W_{nm}. \tag{6.10}$$

The calculation of the signal $x(n)$ can now be naturally considered within the graph framework, where W_{nm} are edge weights.

The probabilities at the stage $(p+1)$ of the random walk transition are calculated starting from the probabilities at the previous stage as in (6.10), which, in the compact matrix form, is given by

$$\mathbf{x}_{p+1} = \mathbf{W}\mathbf{D}^{-1}\mathbf{x}_p$$

or

$$\mathbf{D}^{-1/2}\mathbf{x}_{p+1} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}\mathbf{D}^{-1/2}\mathbf{x}_p,$$

where the matrix $\mathbf{W}$ is a matrix of weighting coefficients and $\mathbf{D}$ is the degree matrix.

In the steady state, when $\mathbf{x}_{p+1} = \mathbf{x}_p = \mathbf{x}$, we have

$$\mathbf{y} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}\mathbf{y}$$

where $\mathbf{y} = \mathbf{D}^{-1/2}\mathbf{x}$. The solution is the smoothest eigenvector of the normalized Laplacian, $\mathbf{L}_N = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$, calculated from

$$(\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2})\mathbf{y} = \mathbf{0},$$

and is given by $\mathbf{y} = [1, 1, \ldots, 1]^T/\sqrt{N}$ or

$$\mathbf{x} = \mathbf{D}^{1/2}[1, 1, \ldots, 1]^T/\sqrt{N}.$$

Note that the vector $\mathbf{x}$ is not constant, and its elements are given by $x(n) = \sqrt{d_n/N}$.

In the **edge-centric random walk**, the probability, p_{nm}, is defined by

$$x_{p+1}(m) = \sum_n x_p(n)p_{nm} = \frac{1}{d_m}\sum_n x_p(n)W_{nm}. \tag{6.11}$$

In this case, the in-flow probability $\sum_n x_p(n)W_{nm}$ for the vertex m is equal (balanced) to the out-flow probability of this vertex, $x_{p+1}(m)d_m = \sum_n x_{p+1}(m)W_{nm}$. This model of random walk is also called the *fluid model* and it has a simple interpretation within the electric circuits framework, since the probabilities (if considered as the electric potentials) satisfy the first Kirchoff low for the vertex m serving as an electric circuit node, that is

$$\sum_n (x_{p+1}(m) - x_p(n))W_{nm} = 0.$$

The matrix form of the edge-centric random walk is given by

$$\mathbf{x}_{p+1} = \mathbf{D}^{-1}\mathbf{W}\mathbf{x}_p$$

or $\mathbf{D}\mathbf{x}_{p+1} = \mathbf{W}\mathbf{x}_p$. In the steady state, for $\mathbf{x}_{p+1} = \mathbf{x}_p = \mathbf{x}$, we have

$$\mathbf{D}\mathbf{x} = \mathbf{W}\mathbf{x}$$

or

$$\mathbf{L}\mathbf{x} = \mathbf{0}. \tag{6.12}$$

The solution to this equation is the smoothest (constant) eigenvector of the graph Laplacian, $\mathbf{x} = [1, 1, \ldots, 1]^T/\sqrt{N}$.

The presented graph theory framework admits for various problem formulations and solutions.

Example 26: Consider the graph from Figure 2.2 in Part I and the case where we desire to find the probabilities, $x(n)$, that the walker reaches vertex 5 before it reaches vertex 7, starting from any vertex n, and assuming that transition probabilities may be defined according the edge-centered random walk model. We therefore have to solve the system $\mathbf{L}\mathbf{x} = \mathbf{0}$, with $x(5) = 1$ and $x(7) = 0$.

In the same way, we can solve another practically interesting problem. A piece of information has reached a member of the social network in Figure 6.10 at vertex 4 (in green circle), but it has not reached the member at vertex 3 (in red circle). The task is to find probabilities that the information is known to a vertex n.

Since the information is present at vertex 4, then $x(4) = 1$ is a certain event, and the fact that the information has not reached vertex 3 means that $x(3) = 0$. Again, according to the analysis from (6.9) to (6.12), we have to solve the system $\mathbf{L}\mathbf{x} = \mathbf{0}$, with $x(4) = 1$ and $x(3) = 0$, that is

$$\begin{bmatrix} 3 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 & 0 & 0 & 0 \\ -1 & 0 & -1 & 3 & 0 & 0 & -1 & 0 \\ 0 & -1 & -1 & 0 & 4 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & 3 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ 0 \\ 1 \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} = \mathbf{0}, \tag{6.13}$$

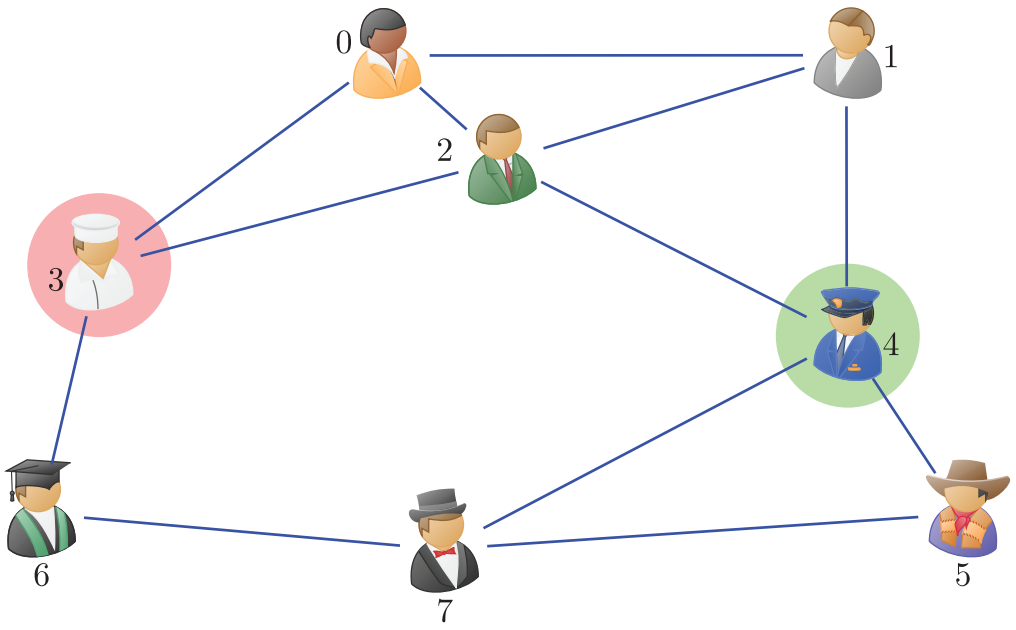


Figure 6.10: A small social network from Figure 6.8, where we are interested in the probability that a piece of news has reached vertex 4 (policeman, in green circle), but has not reached vertex 3 (chef, in red circle). Example 26 considers this scenario within the framework of random walk on graphs.

where the corresponding columns and rows are removed (rows for the known signal values, $x(3)$ and $x(4)$, and the column for the zero-valued signal, $x(3)$), while the green font designates the column to be moved on the right side of the equation for the known signal value, $x(4) = 1$. The solution is obtained from

$$\begin{bmatrix} 3 & -1 & -1 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & 0 \\ -1 & -1 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & -1 \\ 0 & 0 & 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 3 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \tag{6.14}$$

with the inserted values $x(4) = 1$ and $x(3) = 0$, in the following form

$$\mathbf{x} = [0.375,\ 0.625,\ 0.5,\ 0,\ 1,\ 0.875,\ 0.375,\ 0.75]^T.$$

This means that the information is most probably available to the vertex 5, with probability $x(5) = 0.875$, while the lowest probability is that the information is available to the vertices 0 or 6, with probability

$x(0) = x(6) = 0.375$, as can be expected from an intuitive analysis of this graph with a small number of vertices.

6.7 Hitting and Commute Time

The random walk problem is closely related to the hitting and commute time. The *hitting time*, $h(m, n)$, from a vertex m to any vertex n is defined as the expected number of steps for a random walker to travel from the vertex m to a vertex n. Denote by $x_p^{(m)}(l)$ the hitting time from the reference vertex m to the vertices l which are the neighboring vertices of the considered vertex n. Then, the random walker will arrive from a vertex l to the vertex n in one step with the probability that it chooses to transit from the specific l to the considered n. The probability that a random walker is at the neighboring vertex l, and then transits to the vertex, n, is given by

$$p_{ln} = \frac{W_{ln}}{\sum_k W_{nk}} = \frac{1}{d_n} W_{ln}.$$

The hitting time for a vertex n is equal to the sum of all hitting times of neighboring vertices, with one step added, that is

$$x_{p+1}^{(m)}(n) = \sum_l x_p^{(m)}(l) p_{ln} + 1 = \frac{1}{d_n} \sum_l x_p^{(m)}(l) W_{ln} + 1.$$

The matrix form of this equation is

$$\mathbf{x}_{p+1}^{(m)} = \mathbf{D}^{-1}\mathbf{W}\mathbf{x}_p^{(m)} + \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

In the steady state, we have

$$\mathbf{D}\mathbf{x}^{(m)} = \mathbf{W}\mathbf{x}^{(m)} + \mathbf{d},$$

where $\mathbf{d} = \mathbf{D}[1, 1, \ldots, 1]^T$ is a degree vector. Finally the hitting time, $h(m, n) = x^{(m)}(n)$, is a solution to the linear system of equations

$$\mathbf{L}_m \mathbf{x}^{(m)} = \mathbf{d} \tag{6.15}$$

with the reference vertex m, where $x(m) = 0$ is removed from the vector $\mathbf{x}$ to form $\mathbf{x}^{(m)}$ with elements $h(m,n)$, $n = 0, 1, \ldots, N-1$, $n \neq m$. The equation for vertex m is also removed, so that the system is of an $(N-1)$-order and the matrix $\mathbf{L}_m$ is obtained from the graph Laplacian, $\mathbf{L}$, by removing its mth row and mth column.

Example 27: We shall calculate the hitting time for all vertices, n, from the vertex $m = 3$ for the graph from Figure 2.2 in Part I. For this graph, we have

$$\begin{bmatrix} 1.21 & -0.23 & -0.74 & 0 & 0 & 0 & 0 \\ -0.23 & 0.81 & -0.35 & -0.23 & 0 & 0 & 0 \\ -0.74 & -0.35 & 1.59 & -0.24 & 0 & 0 & 0 \\ 0 & -0.23 & -0.24 & 1.12 & -0.51 & 0 & -0.14 \\ 0 & 0 & 0 & -0.51 & 0.66 & 0 & -0.15 \\ 0 & 0 & 0 & 0 & 0 & 0.64 & -0.32 \\ 0 & 0 & 0 & -0.14 & -0.15 & -0.32 & 0.61 \end{bmatrix} \begin{bmatrix} h(3,0) \\ h(3,1) \\ h(3,2) \\ h(3,4) \\ h(3,5) \\ h(3,6) \\ h(3,7) \end{bmatrix} = \begin{bmatrix} 1.21 \\ 0.81 \\ 1.59 \\ 1.12 \\ 0.66 \\ 0.64 \\ 0.61 \end{bmatrix}$$

and this matrix is obtained from the graph Laplacian by removing the row and column corresponding to $m = 3$. The hitting times from the vertex $m = 3$ are then obtained as

$$\begin{bmatrix} h(3,0) \\ h(3,1) \\ h(3,2) \\ h(3,4) \\ h(3,5) \\ h(3,6) \\ h(3,7) \end{bmatrix} = \begin{bmatrix} 9.0155 \\ 11.3003 \\ 9.5942 \\ 12.6594 \\ 13.1427 \\ 6.1930 \\ 10.3860 \end{bmatrix}.$$

The *commute time*, $CT(m,n)$ between vertices m and n is defined as the expected time for the random walker to reach a vertex n starting from vertex m, and then to return (see Part I, Section 4.5), to give

$$CT(m,n) = h(m,n) + h(n,m).$$

Example 28: We consider the task of finding the commute time between the vertices $m = 0$ and $n = N - 1 = 7$ for the graph from Figure 2 in Part I, also shown in Figure 6.11. If we desire to use the full Laplacian matrix and the electric circuit framework for the calculation of the hitting time, then we should include the mth equation with

$h(m,m) = x(m) = 0$. Since the sum of all external sources (on the right side of the Equation (6.15)) must be zero, this means that for the vertex $m = 0$, the right side terms should be $(d_0 - D)$, and the full Laplacian form of (6.15) for the vertex $m = 0$ becomes

$$\mathbf{L}\begin{bmatrix} 0 \\ h(0,1) \\ \vdots \\ h(0,6) \\ h(0,7) \end{bmatrix} = \begin{bmatrix} d_0 - D \\ d_1 \\ \vdots \\ d_6 \\ d_7 \end{bmatrix},$$

where $D = \sum_{i=0}^{N-1} d_i$, and $d_i = \sum_n W_{in}$ are the degrees of vertices, i.

The same relation can be written for $m = 7$ (or any other vertex m), to yield

$$\mathbf{L}\begin{bmatrix} h(7,0) \\ h(7,1) \\ \vdots \\ h(7,6) \\ 0 \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_6 \\ d_7 - D \end{bmatrix}.$$

The difference between the two previous systems of equations is

$$\mathbf{L}\begin{bmatrix} -h(7,0) \\ h(0,1) - h(7,1) \\ \vdots \\ h(0,6) - h(7,6) \\ h(0,7) \end{bmatrix} = \begin{bmatrix} -D \\ 0 \\ \vdots \\ 0 \\ D \end{bmatrix}.$$

This system can be interpreted within the electric circuit framework as the electric circuit with an external source at $m = 0$ whose current is $i(0) = -D$. This external source is closed at $m = 7$ with the current $i(7) = D$, while there are no sources at any other vertex. The difference of voltages in this electric circuit at $m = 0$ and $m = 7$ is equal to the difference of the seventh element, $h(0,7)$, and the first element, $-h(0,7)$, to yield

$$\begin{aligned} x_{0,7} &= h(0,7) - (-h(7,0)) = h(0,7) + h(7,0) \\ &= CT(0,7) = R_{\text{eff}}^{(0,7)} i(7) \end{aligned}$$

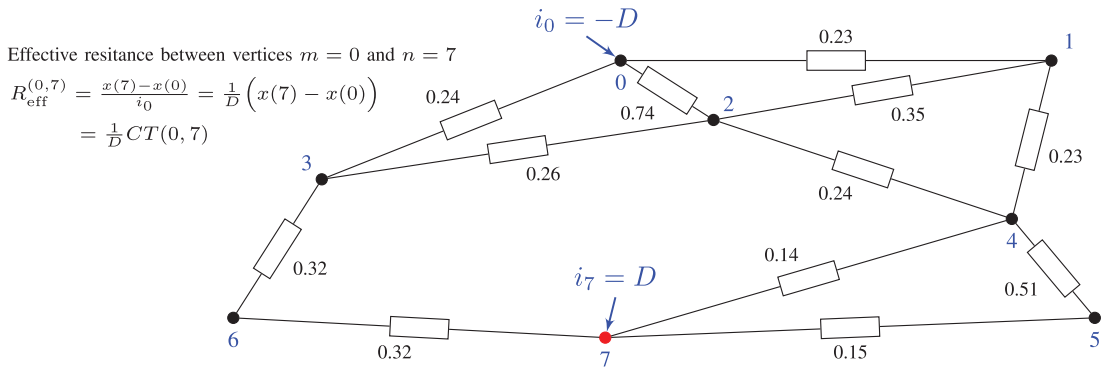


Figure 6.11: Electric circuit interpretation of the commute time, $CT(m,n) = DR_{\text{eff}}^{(m,n)}$.

where $R_{\text{eff}}^{(0,7)}$ is the effective electric resistance between $m = 0$ and $m = N - 1 = 7$, as illustrated in Figure 6.11.

Finally, the previous relation holds for any two vertices, m and n, that is

$$CT(m,n) = DR_{\text{eff}}^{(m,n)}$$

where $D = \sum_{i=0}^{N-1} d_i$.

Example 29: The commute time between vertices $m = 0$ and $n = 7$ for the graph from Figure 2.2 in Part I, also shown in Figure 6.11, can be obtained by calculating the hitting times $h(0,7)$ and $h(7,0)$, as in Example 27. The result is

$$CT(7,0) = h(0,7) + h(7,0) = 10.7436 + 19.6524 = 30.3960.$$

The same result can be obtained by finding the effective resistance between vertices $m = 0$ and $n = 7$ in the electric circuit from Figure 6.11 using the elementary calculations for the effective resistance, $R_{\text{eff}}^{(7,0)}$, to give

$$R_{\text{eff}}^{(7,0)} = 4.0745.$$

With $D = \sum_{i=0}^{7} d_i = 7.46$, the commute time, $CT(7,0) = DR_{\text{eff}}^{(7,0)} = 30.3960$, follows.

6.8 Relating Gaussian Random Signal to Electric Circuits

Consider a random graph signal, $x(n)$, and assume that each sample is Gaussian distributed with mean, μ_n, and standard deviation, σ_n.

Assuming that the signal values are correlated, the pdf of the signal $\mathbf{x}$ is given by

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N}} \sqrt{\det(\boldsymbol{\Sigma}_x^{-1})} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}_x^{-1}(\mathbf{x}-\boldsymbol{\mu})\right). \quad (6.16)$$

The inverse of the autocovariance matrix is the precision matrix $\mathbf{Q} = \boldsymbol{\Sigma}_x^{-1}$. Note that the term precision comes from the one-dimensional case where the precision is inversely proportional to the variance, that is $Q = 1/\sigma^2$.

The maximum likelihood estimate of $\mathbf{x}$ is then obtained from (6.16) by minimizing

$$E_x = \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}_x^{-1}(\mathbf{x}-\boldsymbol{\mu})$$

and the solution is

$$\boldsymbol{\Sigma}_x^{-1}(\mathbf{x}-\boldsymbol{\mu}) = \mathbf{0}. \quad (6.17)$$

For a zero-mean random signal, $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma}_x^{-1}\mathbf{x} = \mathbf{0}$, and the solution in (6.17) corresponds to minimizing the energy of the change (maximal smoothness) in the graph.

The generalized Laplacian corresponding to the precision matrix is defined by

$$\boldsymbol{\Sigma}_x^{-1} = \mathbf{Q} = \mathbf{L} + \mathbf{P}$$

where $\mathbf{P}$ is a diagonal matrix such that the sum of columns of the Laplacian is zero.

The edge weights can now be extracted from the Laplacian matrix. Since the Laplacian is defined using the observed graph signal values, this is a point where the presented analysis meets the discussion from the previous section (see also Examples 10 and 11). The electric circuit form of the minimization condition is obtained from

$$(\mathbf{L}+\mathbf{P})(\mathbf{x}-\boldsymbol{\mu}) = \mathbf{0}$$

or

$$\mathbf{L}\mathbf{x} = -\mathbf{P}\mathbf{x} + (\mathbf{L}+\mathbf{P})\boldsymbol{\mu}.$$

In terms of the external current generators, we can define the problem as

$$\mathbf{L}\mathbf{x} = \mathbf{i}_x + \mathbf{i}_g,$$

where $\mathbf{i}_x = -\mathbf{P}\mathbf{x}$ are voltage-driven current generators and $\mathbf{i}_g = (\mathbf{L}+\mathbf{P})\boldsymbol{\mu} = \mathbf{Q}\boldsymbol{\mu}$ are constant external current generators. Therefore, the steady-state solution can be interpreted and solved in the same way as we solve the described electric circuit. For example, if the observed state is $x(7) = 1$ and $\mu(n) = 0$, we can solve the system for other values of $x(n)$ for a given matrix $\boldsymbol{\Sigma}_x^{-1} = \mathbf{Q} = \mathbf{L} + \mathbf{P}$.

7

Graph Learning from Data and External Sources

In the seminal work on graph topologies by Gabriel Kron, all vertices are assumed to be separable into two groups: *active vertices and inner vertices*. Active vertices are exposed to external sources which may be of different natures, depending on the physical system represented by a graph. Scenarios which admit the grouping into the active and inner vertices include the following.

- In a graph representing an electric circuit, active vertices are characterized by external currents/voltages which supply the network with energy, which is consumed in its resistive edges.
- For graphs representing heat transfer, active vertices must include sources/sinks of heat energy.
- In a transportation network, active vertices are those stations where new influx of passengers can be generated, in contrast with e.g., the transition hubs, where the passengers can only change the lines, and cannot exit or enter the station.
- Active and inner vertices can also be recognized in the postal service network. Here, the external vertices are the points where

the mail is accepted and/or delivered from/to outside world, while the inner vertices are places where the mail is only in transit (sorted and redirected).

- In a computer network, the inner vertices are the servers with no external input/output function, but only have the store and data transfer functionality. Computers that can generate input or output data would be active vertices (vertices with external sources).

Notice that the external sources drive the system (graph), while the inner sources can be organized in various ways to improve the efficiency of the system. In the Kron reduction of graphs, inner vertices are commonly reorganized by appropriate transformations of graphs, such as the "star-mesh" transformations.

In the previous section, learning of graph topology from data on the graph has been considered based on the correlation and precision matrices. The fundamental additional assumption is that the graph signal is smooth over the vertices. Notice that if we were able to measure the graph signal and external sources in the active vertices, then this would make it possible to learn graph topology in an exact way.

Consider the pth observation of a signal on a graph,

$$\mathbf{x}_p = [x_p(0), x_p(1), \ldots, x_p(N-1)]^T,$$

and the corresponding external sources,

$$\mathbf{i}_p = [i_p(0), i_p(1), \ldots, i_p(N-1)]^T.$$

Without loss of generality, assume that the $(N-1)$th vertex is a reference, where $x_p(N-1) = 0$ and $i_p(N-1) = -\sum_{n=0}^{N-2} i_p(n)$. These elements will be removed from the data and equations and only the data on remaining vertices will be considered, and denoted as $\mathbf{x}_p = [x_p(0), x_p(1), \ldots, x_p(N-2)]^T$ and $\mathbf{i}_p = [i_p(0), i_p(1), \ldots, i_p(N-2)]^T$. The electric circuit equation in (6.1) for the reduced set of data is then

$$\begin{bmatrix} \boldsymbol{l}_0 \\ \boldsymbol{l}_1 \\ \vdots \\ \boldsymbol{l}_{N-2} \end{bmatrix} \mathbf{x}_p = \mathbf{i}_p,$$

where

$$\boldsymbol{l}_i = [L_{i0}, L_{i1}, \ldots, L_{i,N-2}]$$

are the rows of the graph Laplacian matrix, $\mathbf{L}$, with the elements ranging from $n = 0$ to $n = N - 2$. In other words, the last element and the last row in the graph Laplacian, which correspond to the reference vertex, $n = N - 1$, are omitted.

If P sets of observations are available, then we can write this system in the form

$$\begin{bmatrix} \boldsymbol{l}_0 \\ \boldsymbol{l}_1 \\ \vdots \\ \boldsymbol{l}_{N-2} \end{bmatrix} [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P] = [\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_P].$$

or

$$\begin{bmatrix} \boldsymbol{l}_0 \\ \boldsymbol{l}_1 \\ \vdots \\ \boldsymbol{l}_{N-2} \end{bmatrix} \mathbf{X}_{N-1,P} = \mathbf{J}_{N-1,P}$$

The matrices $\mathbf{X}_{N-1,P}$ and $\mathbf{J}_{N-1,P}$ represent respectively the signal on the graph and external source matrices of dimensionality $(N-1) \times P$.

Now, we can consider the following cases.

- With $P \geq N - 1$ independent available observations, the exact form of the graph Laplacian (its first $(N-1)$ rows and columns) follows from

$$\begin{bmatrix} \boldsymbol{l}_0 \\ \boldsymbol{l}_1 \\ \vdots \\ \boldsymbol{l}_{N-2} \end{bmatrix} = \mathbf{J}_{N-1,P} \ \text{pinv}\{\mathbf{X}_{N-1,P}\}. \tag{7.1}$$

  The last column and the last row of the graph Laplacian, $\mathbf{L}$, are formed in such a way that the sum over every column or row is zero.

- A more complex case arises when $P < N - 1$. Then, the number of observations is not sufficient to recover the graph Laplacian.

However, if we assume that the graph Laplacian is sparse, with a small number of nonzero elements (edges), the solution is achievable within the compressive sensing framework. In order to adapt the system in (7.1) to suit the standard LASSO algorithm, we shall rewrite it in the form

$$\mathbf{X}_{N-1,P}^{T} \begin{bmatrix} \boldsymbol{l}_0 \\ \boldsymbol{l}_1 \\ \vdots \\ \boldsymbol{l}_{N-2} \end{bmatrix}^{T} = \mathbf{J}_{N-1,P}^{T}.$$

Now, LASSO minimization can be performed for each data column, $\boldsymbol{l}_k^T$, and the corresponding column of the external source matrix $\mathbf{J}_{N-1,P}^T$, denoted by $\boldsymbol{i}_k$, in the form

$$\boldsymbol{l}_k = \text{lasso}(\mathbf{X}_{N-1,P}^T, \boldsymbol{i}_k, \rho).$$

Another approach would be to transform the matrices with graph Laplacian rows, $\boldsymbol{i}_k$, and external source matrix, $\mathbf{J}_{N-1,P}^T$, into column vectors to have

$$\begin{bmatrix} \mathbf{X}_{N-1,P}^T & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{X}_{N-1,P}^T & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{X}_{N-1,P}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{l}_0^T \\ \boldsymbol{l}_1^T \\ \vdots \\ \boldsymbol{l}_{N-2}^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{i}_0 \\ \boldsymbol{i}_1 \\ \vdots \\ \boldsymbol{i}_{N-2} \end{bmatrix}.$$

Using the notation

$$(\mathbf{I}_{N-1,N-1} \otimes \mathbf{X}_{N-1,P}^T)\, \boldsymbol{l}_{\text{vec}} = \boldsymbol{i}_{\text{vec}}$$

for the matrix and the vectors in the above equation (where $\mathbf{I}_{N-1,N-1}$ is the identity matrix), this system can be solved using

$$\boldsymbol{l}_{\text{vec}} = \text{lasso}(\mathbf{I}_{N-1,N-1} \otimes \mathbf{X}_{N-1,P}^T, \boldsymbol{i}_{\text{vec}}, \rho).$$

Example 30: Consider a graph with $N = 50$ vertices, and with a small number of edges. Such a sparse graph Laplacian, $\mathbf{L}$, is shown in Figure 7.1(a). The graph Laplacian was estimated using a large number, $P = 60$, of observations of the graph signal and external sources. Both

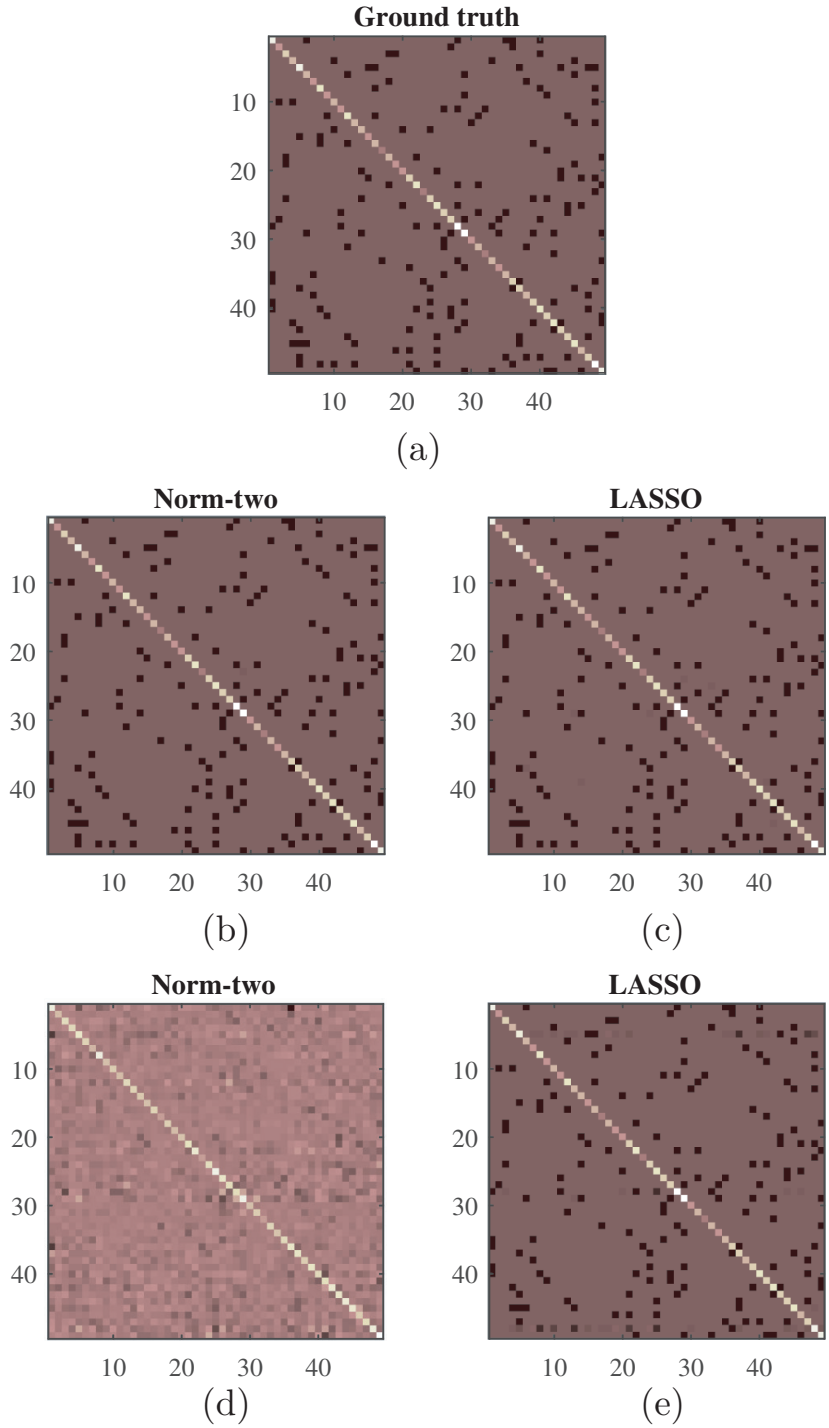


Figure 7.1: Estimation of the graph Laplacian, $\mathbf{L}$, for a graph with $N = 50$ randomly positioned vertices and a small number of edges. (a) Ground truth graph Laplacian, $\mathbf{L}$. (b) Estimated graph Laplacian using the norm-two for a large number of observations, $P = 60 > N = 50$. (c) Estimated graph Laplacian using the LASSO, for a large number of observations, $P = 60 > N = 50$. (d) Estimated graph Laplacian using the norm-two, for a small number of observations, $P = 40 < N = 50$. (e) Estimated graph Laplacian using the LASSO, for a small number of observations, $P = 40 < N = 50$.

the norm-two and the LASSO estimates of the graph Laplacian were accurate, as shown in Figures 7.1(b) and (c). Next, the number of observations was reduced to $P = 40 < N = 50$. In this case, the sparsity of the graph Laplacian is crucial for the solution. The LASSO algorithm, which includes the sparsity constraint, was still able to produce a good result, as seen from Figure 7.1(e). The norm-two was calculated using the pseudo-inverse of the data matrix, $\mathbf{X}_{N-1,P}^T$, and was not appropriate as the graph Laplacian estimator, as seen in Figure 7.1(d).

Finally, we should mention that in Example 30 we have not used the conditions that the graph Laplacian is a symmetric matrix and that the elements of the weight matrix, W_{mn}, from which the graph Laplacian elements are formed, are nonnegative. These conditions can be used within linear programming formulations to improve the estimation.

8

Random Signal Simulation on Graphs

This section addresses ways to simulate graph signals, as for testing of any new method for learning graph topology based on the available data, we have to show that the method is reliable and accurate on simulated graphs and graph signals. To this end, we have to assume a graph (randomly structured) and then to simulate data on such a graph. Such data should exhibit a desired degree of randomness, in order to infer the influence of graph connections to the signal form. Notice that a graph signal should be influenced by the structure of a graph in an implicit way. This influence is then used as the basis for graph topology learning. Graph signal simulation is not straightforward and certainly not a unique process. Some of the presented approaches to graph signal simulation are based on the previously introduced analysis of signals on graphs with physically well-defined topology in Section 6.

The representation of a graph and graph signal within the circuit theory framework can be used to simulate random signals on graphs. While several approaches are possible, we will here present some of the most frequently used ones.

(1) Assume that the graph is initiated by **external sources** that are random variables. In that case, the pth observation of a random

signal on this graph is simulated as a solution of the system of equations

$$\mathbf{L}\mathbf{x}_p = \boldsymbol{\varepsilon}_p,$$

with $\mathbf{i}_p = \boldsymbol{\varepsilon}_p$. Note that one of the external sources (randomly chosen for each observation p) should compensate for all other sources, to ensure $\sum_{n=0}^{N-1} \varepsilon_p(n) = 0$.

Since the graph Laplacian matrix is singular, the graph signal value (the electric potential in the electric circuit case) at a vertex, for example, for $n = 0$, should be considered as *a reference* and its value should be assumed, for example, $x(0) = 0$. This strategy may be used whenever the inversion of the graph Laplacian is required.

(2) The graph is **excited at only one of its vertices** (and one additional reference vertex) with a random external zero-mean white source. The position of these vertices is randomly selected for each p. Then, the random signal observation on a graph is obtained as a solution to

$$\mathbf{L}\mathbf{x}_p = \mathbf{i}_p$$

where $i_p(n) = \varepsilon_p\delta(n - n_i) - \varepsilon_p\delta(n - n_j)$ and n_i and n_j are two randomly selected vertices at each observation.

(3) A minimal information needed to calculate a random graph signal is the knowledge of signal values at two randomly positioned vertices. Assuming that $x_p(n) = \varepsilon_p\delta(n - n_i) + \epsilon_p\delta(n - n_j)$ and n_i and n_j are two randomly selected vertices at each observation; then we may solve the system for all other signal samples, based on

$$\mathbf{L}\mathbf{x}_p = \mathbf{0}.$$

With the two assumed values, $x_p(n)$, at $n = n_i$ and $n = n_j$, we can solve this system for all other signal values. In the case of external sources, the values should be compensated (their sum should be zero), as mentioned earlier. In this case, there is no need for compensation, which means that ε_p and ϵ_p could be independent random variables.

(4) The signal on a graph is formed using a linear combination of white noise $\boldsymbol{\varepsilon}_p$ and its graph shifted versions. The output signal after M such graph shifts, defined by the normalized Laplacian, is given by

$$\mathbf{x}_p = (h_M \mathbf{L}^M + h_{M-1}\mathbf{L}^{M-1} + \cdots + h_1 \mathbf{L}^1 + h_0 \mathbf{L}^0)\boldsymbol{\varepsilon}_p. \qquad (8.1)$$

The resulting graph signal can be written in the form

$$\mathbf{x}_p = H(\mathbf{L})\boldsymbol{\varepsilon}_p.$$

(5) Analysis based on the adjacency matrix and graph shifts. Assume that an undirected graph with the adjacency matrix $\mathbf{A}$, is excited at N_a randomly chosen vertices $n_1, n_2, \ldots, n_{N_a}$, $\eta = N_a/N$, with spikes $\delta(n - n_i)$, $i = 1, 2, \ldots, N_a$. After shifting these spikes K times, we obtain

$$\mathbf{x} = \mathbf{A}^K \sum_{i=1}^{N_a} \boldsymbol{\delta}_{n_i},$$

where $\boldsymbol{\delta}_{n_i}$ is a graph signal with a nonzero value at $n = n_i$ only. The parameters K and N_a define the resulting signal smoothness. An example of one realization of such a signal is presented in Part II, Figure 3.6 for $\eta = 1/8$, $K = 1$ (upper subplots) and $\eta = 2/8$, $K = 1$ (lower subplots) using the spikes $a_i\delta(n - n_i)$, where a_i are the spike amplitudes.

(6) Signals are commonly simulated as sums of the harmonic basis functions, as in classical Fourier analysis. This kind of simulation may be used in graph signal processing, too. Such a signal on a graph can be written as

$$\mathbf{x} = \sum_{i=1}^{K} a_{k_i} \mathbf{u}_{k_i}$$

where $\mathbf{u}_k$ are the eigenvectors of the Laplacian or adjacency matrix, and a_k are random constants. This kind of graph signal simulation, with or without an additive noise, has been often used in this part.

9

Summary of Graph Learning from Data Using Probabilistic Generative Models

Analytics of data on graphs with known or given topologies are feasible for applications that involve physically meaningful structures, such as citation networks, transport networks and observable social networks. In those applications, various vertex or spectral domain techniques, as mentioned in Part II of this monograph, have been successfully implemented and developed to filter, analyze or visualize graph signals. However, in many situations where the graph topology cannot be directly observed or even when the data is partially observed, the inference of graph structure is a key first step.

Remark 13: Given the observed graph data, graph topology learning is an ill-posed problem. In other words, totally different graphs can generate the same data, while one set of observed data can result in different graph topologies, depending on the graph learning method used. Thus, to infer graph topology we need to employ some priors, for example, to match statistics by imposing sparsity or smoothness conditions on the graph.

Previous sections in this monograph have introduced various sparsity promotion techniques, such as the graphical LASSO (GLASSO) and smoothness constrained graph learning, mostly from the perspective

of linear algebra (Dong *et al.*, 2019; Giannakis *et al.*, 2018; Mateos *et al.*, 2019). However, it is more natural to connect and summarise such techniques under the umbrella of probabilistic generative models. A straightforward approach would be on the basis of some fundamental statistical models, such as the covariance or precision matrices of Gaussian distribution (due to their positive definiteness property), the Gaussian Markov random fields with local independence prior, or a factor analysis models with smoothness assumption. We also envisage further progress of generative models to be based on the concept of diffusion processes on graphs, whereby the signal generating process can be regarded as the graph signal that has been diffused by some graph kernels (for example, polynomial kernels) from a white Gaussian distributed noise.

Generally speaking, graph learning can be treated as an inverse operation to the graph data generation process, that is, $\mathbf{x} = f_{\mathcal{G}}(\mathbf{z})$, where $\mathbf{x}$ denotes the observed data. Here, the data are considered to be the output of an unknown transform (denoted by $f_{\mathcal{G}}$) of some initial state, $\mathbf{z}$, on the graph $\mathcal{G}$. Existing literature on learning a graph can be thought of as an attempt to infer the generative process, $f_{\mathcal{G}}$, by matching the data statistics, $\mathbf{x}$, with different priors that are imposed on $\mathbf{z}$. We should point out that in this section we discuss the problem of learning graphs from fully observed graph data because of its underpinning role in a number of advanced techniques, such as graph learning with partially observed data (Grotas *et al.*, 2019; Wai *et al.*, 2019) and dynamic graph learning (Chen *et al.*, 2011; Ioannidis *et al.*, 2019b; Kaplan, 2008).

9.1 Basic Gaussian Models

The simplest way of constructing a graph would be to associate edge weights with the covariance of data observed on a graph; this is reasonable under the Gaussian assumption, since the first two moments fully capture the whole statistics of the distribution. Indeed, the non-zero elements of the covariance matrix of graph data naturally provide consistent estimation of the connectivity within a graph. This method is explained within the introductory part of Section 4.

Given a set of P independent and identically distributed (i.i.d.) observed data vectors, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P$, the empirical sample covariance is calculated as

$$\Sigma_x(m,n) = \frac{1}{P}\sum_{p=1}^{P}(x_p(m) - \mu(m))(x_p(n) - \mu(n)), \tag{9.1}$$

or

$$\mathbf{\Sigma} = \frac{1}{P}\sum_{p=1}^{P}(\mathbf{x}_p - \overline{\mathbf{x}})(\mathbf{x}_p - \overline{\mathbf{x}})^T, \tag{9.2}$$

where $\overline{\mathbf{x}}$ is the mean value of the observed samples. Alternatively, a normalized version of $\mathbf{\Sigma}$ can also be employed in order to produce the edge weights as

$$\sigma_x(m,n) = \frac{\Sigma_x(m,n)}{\sqrt{\Sigma_x(m,m)\Sigma_x(n,n)}}. \tag{9.3}$$

For this empirical statistics, we can employ a threshold, τ, to designate the non-zero connections of the adjacency weight, in a way similar to (2.1), to yield

$$W_{m,n} = \begin{cases} \sigma_x(m,n), & |\sigma_x(m,n)| \geq \tau \\ 0, & |\sigma_x(m,n)| < \tau. \end{cases} \tag{9.4}$$

A more sophisticated approach would be to use hypothesis testing via setting a false alarm rate, whereby

$$\mathcal{H}_0\text{: } \sigma_x(m,n) = 0 \text{ versus } \mathcal{H}_1\text{: } \sigma_x(m,n) \neq 0. \tag{9.5}$$

In these scenarios, the empirical covariance is a common choice of test statistics. Although the density of $\sigma_x(m,n)$ may have closed-form representation, it typically needs numerical integration when calculating the p-values; however, transformations of $\sigma_x(m,n)$ can relax this issue to obtain closed-form densities. For example, under the Gaussian distribution and $\mathcal{H}_0$, the weighting

$$s(m,n) = \frac{\sigma_x(m,n)\sqrt{P-2}}{\sqrt{1-\sigma_x^2(m,n)}}$$

would satisfy a student t-distribution of $(P-2)$ degrees of freedom, and

$$s(m,n) = \tanh^{-1}(\sigma_x(m,n))$$

would then result in a Gaussian distribution with zero mean and $1/(P-3)$ variance (see Chapter 7.3.1 Kolaczyk, 2009). In those transformed test statistics, the statistical significance can be easily adjusted to meet the false alarm rate. However, the limitation of this model is that by employing individual tests, the number of implementations in inferring the graph grows as $\mathcal{O}(N^2)$. This is computational prohibitive for relatively large graphs; on the other hand, this leads to increasingly false judgments even with a constant false alarm rate.

A further possible misleading due to the correlation models stems from the fact that the *correlation does not mean the causation*. In other words, the mth and nth vertices can show a strong correlation when they are both highly influenced by an intermediate vertex, however, they are not the causation of one another, as illustrated in Example 5.

9.2 Gaussian Graphical Model

To address the issues with the correlation and causation, and to be able to construct a graph that reflects only direct relationships among its vertices, one classical method employs the partial correlation, whereby the correlation of two vertices is calculated by eliminating associations of other contributing vertices. Under the assumption that vertices satisfy some mild distributions such as elliptical distributions, the partial correlation coincides with the conditional correlation (Baba *et al.*, 2004), and further equals to the conditional independence under the Gaussian assumption on vertices; this allows the partial correlation to be explicitly related to the precision matrix. The so established relationship is crucial in understanding other techniques such as the GLASSO, graph regression and other generative models.

9.2.1 Partial Correlation Model

In order to simplify the notation, and without loss of generality, we shall consider vertices $n = 0$ and $m = 1$. The set of all other vertices, except for the mth and the nth vertex, is denoted by $\mathcal{V}\backslash\{m, n\} = \{2, 3, \ldots, N-1\}$. Define the data vectors at each vertex by $\mathbf{y}_n$, as in (4.1), and denote by $\hat{\mathbf{y}}_0$ and $\hat{\mathbf{y}}_1$ the best linear approximations to the

signal samples $\mathbf{y}_0$ and $\mathbf{y}_1$, obtained based on the data at other vertices, $\mathbf{y}_2, \mathbf{y}_3, \ldots, \mathbf{y}_{N-1}$. The new data values are then defined as

$$\mathbf{z}_0 = \mathbf{y}_0 - \hat{\mathbf{y}}_0$$
$$\mathbf{z}_1 = \mathbf{y}_1 - \hat{\mathbf{y}}_1.$$

Now, the (empirical) partial correlation between the vertices $m = 0$ and $n = 1$ can be defined as

$$\sigma_z(0,1) = \frac{\Sigma_z(0,1)}{\sqrt{\Sigma_z(0,0)}\sqrt{\Sigma_z(1,1)}}. \tag{9.6}$$

In a similar way, all other partial correlations, $\sigma_z(m,n)$, between pairs of vertices m and n are calculated. Then, one way of hypothesis testing is as follows

$$\mathcal{H}_0\text{: } \sigma_z(m,n) = 0 \text{ versus } \mathcal{H}_1\text{: } \sigma_z(m,n) \neq 0, \tag{9.7}$$

where $\sigma_z(m,n)$ is employed as the test statistics. Other choices of test statistics, such as the Fisher's transform $s(m,n) = \tanh^{-1}(\sigma_z(m,n))$, also obtain an asymptotically Gaussian distribution (Chapter 7.3.2 Kolaczyk, 2009).

9.2.2 Gaussian Markov Random Field

A further assumption which can be imposed on the partial correlation model is that of the Gaussian distribution, which in many cases is a common setting as this facilitates closed-form solutions and ease of analysis. For example, under the Gaussian distribution, the partial correlation coincides with the conditional correlation (Baba *et al.*, 2004), or equivalently, conditional independence; this in turn forms the pairwise Markov property of random fields, which constitutes a Gaussian Markov random field.

We shall denote the mth and the nth elements of graph signal samples by $\mathbf{y}_A$, and all other elements except for the mth and the nth element by $\mathbf{y}_B$. The covariance of $\mathbf{y}_A$ is then designated by $\mathbf{\Sigma}_{AA}$, and is of the size 2×2. The so obtained block structure of (9.2) becomes

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_{AA} & \mathbf{\Sigma}_{AB} \\ \mathbf{\Sigma}_{BA} & \mathbf{\Sigma}_{BB} \end{bmatrix}. \tag{9.8}$$

The covariance of the corresponding $\mathbf{y}_A$ conditioned on $\mathbf{y}_B$ is then easily obtained as

$$\boldsymbol{\Sigma}_{A\,|\,B} = \boldsymbol{\Sigma}_{AA} - \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\boldsymbol{\Sigma}_{BA}, \tag{9.9}$$

which is also called the Schur complement. On the other hand, to rewrite the expression in (9.8) with regard to the precision matrix, $\mathbf{Q} = \boldsymbol{\Sigma}^{-1}$, we can use the following block-wise matrix property

$$\begin{aligned}\mathbf{Q} = \boldsymbol{\Sigma}^{-1} &= \begin{bmatrix} \boldsymbol{\Sigma}_{A\,|\,B}^{-1} & -\boldsymbol{\Sigma}_{A\,|\,B}^{-1}\boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1} \\ -\boldsymbol{\Sigma}_{BB}^{-1}\boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{A\,|\,B}^{-1} & \boldsymbol{\Sigma}_{BB}^{-1}\boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{A\,|\,B}^{-1}\boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{Q}_{AA} & \mathbf{Q}_{AB} \\ \mathbf{Q}_{BA} & \mathbf{Q}_{BB} \end{bmatrix}. \end{aligned} \tag{9.10}$$

From (9.10), observe that $\boldsymbol{\Sigma}_{A\,|\,B} = \mathbf{Q}_{AA}^{-1}$ if the inverse of $\mathbf{Q}_{AA}$ exists. In other words, to obtain the partial correlations in (9.9), it is more convenient to use the precision matrix than the covariance matrix. Thus, one feasible way to associate the edge weights is via

$$W_{m,n} = -\frac{Q(m,n)}{\sqrt{Q(m,m)Q(n,n)}}, \tag{9.11}$$

where $\mathbf{Q} = \boldsymbol{\Sigma}^{-1}$ is the empirical precision matrix. Then, the association of edge weights can be used to infer non-zero elements of $W_{m,n}$, which is also known as the covariance selection problem (Dempster, 1972). One feasible method is to recursively update the graph by testing the hypotheses in the form

$$\mathcal{H}_0\colon W_{m,n} = 0 \text{ versus } \mathcal{H}_1\colon W_{m,n} \neq 0, \tag{9.12}$$

where the $W_{m,n}$ is used as the test statistic. For large-scale graphs, however, this model also shows limitations which are similar to those of correlation models in Section 9.1. Although this model can relax the vagueness regarding the correlation and causation, it has one additional limitation, in that it requires the number of samples to be larger than the dimension of covariance to ensure a proper inverse of covariance; this does not necessarily hold, especially for large-scale graphs, as stated in Remark 8. The graphical LASSO and linear regression methods may be used to solve this issue.

9.2.3 Graphical LASSO and Regression Models

A common way of overcoming the problem of rank deficiency is to involve a regularization term when estimating the precision matrix.

Given the set of independent and identically distributed samples, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P$, the log-likelihood of a Gaussian distribution with zero mean and precision matrix $\mathbf{Q}$ is represented as in (4.19)

$$J = \sum_{p=1}^{P} \Big(-\frac{1}{2}\mathbf{x}_p^T \mathbf{Q}\mathbf{x}_p - \frac{P}{2}\ln(2\pi) + \frac{1}{2}\ln|\mathbf{Q}| \Big) \tag{9.13}$$

$$\propto P\ln|\mathbf{Q}| - \sum_{p=1}^{P}(\mathbf{x}_p^T \mathbf{Q}\mathbf{x}_p), \tag{9.14}$$

where $|\mathbf{Q}| = \det(\mathbf{Q})$. The maximization of this log-likelihood yields the attained optimum in the form

$$\mathbf{Q}^{-1} = \frac{1}{P}\sum_{p=1}^{P}\mathbf{x}_p\mathbf{x}_p^T,$$

as in (4.25)–(4.26).

However, when P is smaller than the dimension of $\mathbf{x}_p$, the term $\sum_{p=1}^{P}(\mathbf{x}_p\mathbf{x}_p^T)$ is not full rank, which causes the singularity of $\mathbf{Q}$. One way of avoiding this issue is to use the l_1 norm to promote sparsity in (9.13), in a similar form to (4.27), to yield

$$\overline{J} = P\ln|\mathbf{Q}| - \sum_{p=1}^{P}(\mathbf{x}_p^T \mathbf{Q}\mathbf{x}_p) - \rho\|\mathbf{Q}\|_1, \tag{9.15}$$

which is known as a GLASSO problem. As shown in Yuan and Lin (2007), in this way the correct graph can be inferred with the probability approaching one, when the parameter ρ is chosen to satisfy $\rho \cdot P \to \infty$ and $\rho \cdot \sqrt{P} \to 0$, for $P \to \infty$.

Remark 14: Apart from the l_1 norm, other regularization strategies can also be employed in (9.15). For example, solving (9.15) could result in negative values, which of course have no meaning when associating the edge weights. Thus, constraining edge weights to be non-negative is also a common regularization approach in graph learning (see also

Section 4.3). For more detail, we refer to Friedman *et al.* (2008), Banerjee *et al.* (2008), and Yuan and Lin (2006).

Graph regression. Another perspective of learning the Gaussian graphical model (described in Section 4.1 and Example 6) is via a regression of data observed at each vertex, $\mathbf{y}_m$, given the data observations at other vertices, $\mathbf{y}_n$, $n \in \{0, 1, 2, \ldots, m-1, m+1, \ldots, N-1\} = \mathcal{V}\backslash\{m\}$. The aim of the regression here is to learn a graph that yields the minimum mean square error, given the observed samples. More specifically, the values β_{nm}, $n = \mathcal{V}\backslash\{m\}$, that minimize

$$J_m = \left\| \mathbf{y}_m - \sum_{n=0, n\neq m}^{N-1} \beta_{nm} \mathbf{y}_n \right\|_2^2 \tag{9.16}$$

follow from

$$\left(\mathbf{y}_m - \sum_{n=0, n\neq m}^{N-1} \beta_{nm} \mathbf{y}_n \right) \mathbf{y}_k^T = \mathbf{0}$$

or $\sum_{n=0, n\neq m}^{N-1} \beta_{nm} \Sigma_x(n,k) = \Sigma_x(m,k)$, for $k, n \in \mathcal{V}\backslash\{m\}$. A matrix solution to this equation is

$$\boldsymbol{\beta}_m = \boldsymbol{\Sigma}_{mm}^{-1} \boldsymbol{\Sigma}_{1m},$$

where $\boldsymbol{\Sigma}_{1m}$ is a vector with $(N-1)$ elements $\Sigma_x(m,k)$, $k = \mathcal{V}\backslash\{m\}$, and $\boldsymbol{\Sigma}_{mm}$ is an $(N-1) \times (N-1)$ matrix with elements $\Sigma_x(n,k)$, $k, n = \mathcal{V}\backslash\{m\}$. On the other hand, under the Gaussian assumption, the conditional mean of $\mathbf{y}_m$ on $\mathbf{y}_n$ is given by

$$\mathrm{E}_{p(\mathbf{y}_m \,|\, \mathbf{y}_n)}\{\mathbf{y}_m\} = (\boldsymbol{\Sigma}_{mm}^{-1} \boldsymbol{\Sigma}_{1m})^T \mathbf{X}_{P,m},$$

where

$$\mathbf{X}_{P,m} = \begin{bmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{m-1} \\ \mathbf{y}_{m+1} \\ \vdots \\ \mathbf{y}_{N-1} \end{bmatrix},$$

with

$$\mathbf{y}_n = [x_1(n), x_2(n), \ldots, x_P(n)]. \tag{9.17}$$

Therefore, to infer $\mathbf{Q}$, given the data observed on a graph, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P$, we can regress x_m for each vertex, m, on the basis of (9.16) as follows,

$$x_m = \boldsymbol{\beta}_m^T \mathbf{X}_{P,m} + \boldsymbol{\epsilon}_m, \tag{9.18}$$

where $\boldsymbol{\epsilon}_m$ is independent Gaussian noise.

Therefore, the problem of learning $\mathbf{Q}$ turns into the regression problem on $\boldsymbol{\beta}_m$, for each vertex, while non-zero elements in $\boldsymbol{\beta}_m$ also indicate the corresponding non-zero elements in $\mathbf{Q}$, namely, the edges in the graph.

The main advantage of regression-style methods is that the regressions for all vertices can be computed in parallel, which provides significantly relaxed computation when learning large graphs. However, additional attention should be paid to the symmetry of the learnt regression coefficients when dealing with an undirected graph, for example as in (4.11), more detail can be found in Meinshausen *et al.* (2006). The condition of coefficient sparsity could also be included, which leads to the LASSO formulation and solution to this problem, as in Section 4.1.

9.2.4 Factor Analysis Model

In Sections 9.1 and 9.2, the Gaussian distribution was assumed and on the basis of this distribution, most methods have been proposed to learn the graph edges in a recursive manner, i.e., by learning an edge per iteration. On the other hand, such methods can be regarded as a generative process via a basic Gaussian distribution, whereby the covariance or the precision matrix is nontrivially associated with the graph edges. It is thus natural to adopt more general and sophisticated models in graph learning.

One important model in probabilistic generative models is the factor analysis model, which forms the basis of many important tools, such as the probabilistic principal component analysis. Therefore, the observed data on a graph, $\mathbf{x}$, is assumed to be generated via a factor model that can be represented as

$$\mathbf{x} = \mathbf{U}\mathbf{v} + \epsilon, \tag{9.19}$$

where $\mathbf{U}$ is a unitary matrix of the graph Laplacian eigenvectors, and $\mathbf{v}$ is a vector of latent variables (or factor loadings) which is Gaussian distributed with zero mean and a diagonal precision matrix corresponding to the graph Laplacian eigenvalues $\mathbf{\Lambda}$, that is,

$$\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Lambda}^{-1}),$$

where $\mathbf{\Lambda}^{-1}$ is the Moore-Penrose pseudoinverse of $\mathbf{\Lambda}$, while $\epsilon \sim \mathcal{N}(\mathbf{0}, \alpha^2\mathbf{I})$ is also Gaussian distributed but independent of latent variables $\mathbf{v}$.

On the basis of this factor model, it is easy to obtain the distribution of the observations, $\mathbf{x}$, as

$$\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{U}\mathbf{\Lambda}^{-1}\mathbf{U}^T + \alpha^2\mathbf{I}).$$

The term $(\mathbf{U}\mathbf{\Lambda}^{-1}\mathbf{U}^T)^{-1} = \mathbf{U}^T\mathbf{\Lambda}\mathbf{U}$ uniquely defines the Laplacian matrix, $\mathbf{L}$, of a graph. This allows us to infer the graph structure by learning $\mathbf{L} = \mathbf{U}^T\mathbf{\Lambda}\mathbf{U}$ from the factor model via maximizing the posterior distribution of $\mathbf{v}$ given $\mathbf{x}$, that is

$$P(\mathbf{v} \,|\, \mathbf{x}) \propto P(\mathbf{x} \,|\, \mathbf{v})P(\mathbf{v}) \propto e^{-\frac{(\mathbf{x}-\mathbf{U}\mathbf{v})^T(\mathbf{x}-\mathbf{U}\mathbf{v})}{\alpha^2}} e^{-\mathbf{v}^T\mathbf{\Lambda}\mathbf{v}}.$$

The log-likelihood form then follows as Dong *et al.* (2016)

$$\min_{\mathbf{\Lambda},\mathbf{U},\mathbf{v}} \|\mathbf{x} - \mathbf{U}\mathbf{v}\|^2 + \rho \cdot \mathbf{v}^T\mathbf{\Lambda}\mathbf{v}, \tag{9.20}$$

where ρ is a hyperparameter that balances between the influence of mean square error $\|\mathbf{x} - \mathbf{U}\mathbf{v}\|^2$ and the positive definiteness constraint $\mathbf{v}^T\mathbf{\Lambda}\mathbf{v}$. Expression (9.20) can be further rewritten using the notation $\mathbf{y} = \mathbf{U}\mathbf{v}$, as

$$\min_{\mathbf{L},\mathbf{y}} \|\mathbf{x} - \mathbf{y}\|^2 + \rho \cdot \mathbf{y}^T\mathbf{L}\mathbf{y}. \tag{9.21}$$

By inspection of (9.21) we see that the term $\mathbf{y}^T\mathbf{L}\mathbf{y}$ measures the smoothness of signal $\mathbf{y}$ on the graph; in other words, (9.21) minimises the distance between the observed samples and the generated signals, whilst imposing the smoothness on the generated signals, as discussed in Section 4.2. Other regularizations can also be imposed onto this model, such as that trace($\mathbf{L}$) is equal to the dimension of the graph, in order to avoid a trivial all zero optimum and non-positive values in the off-diagonal elements of $\mathbf{L}$, and to learn a feasible graph

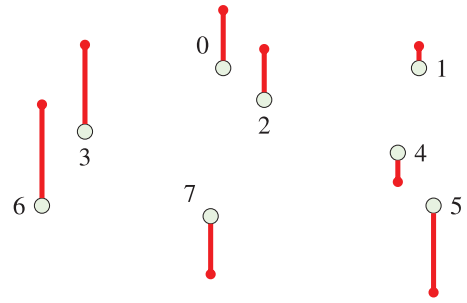
(Dong *et al.*, 2016). Finally, (9.21) can be optimized in an alternative manner, as discussed in Section 4.2 and Algorithm 2, namely, by alternatively optimizing one of the two parameters ($\mathbf{L}$ or $\mathbf{y}$) while fixing the other one.

Further improvements following the factor model of learning a smooth graph include the use of a more flexible smoothness prior when optimizing $\mathbf{L}$ in an alternative optimization, as various constrains on the $\mathbf{L}$ can lead to complicated optimization implementations (Kalofolias, 2016). This is achieved by rewriting the smoothness prior, $\mathbf{y}^T\mathbf{L}\mathbf{y}$ in (9.21), as $\mathbf{y}^T\mathbf{L}\mathbf{y} = \frac{1}{2}\sum_{m,n}\mathbf{A}_{mn}(y(m) - y(n))^2$ so that the constrains can be explicitly imposed on the adjacency matrix $\mathbf{A}$, instead of on the Laplacian $\mathbf{L}$. It is also possible to learn graph by selecting the edges from atoms in a dictionary (called the incidence matrix) (Chepuri *et al.*, 2017). Although this strategy can explicitly control the sparsity of the graph, it cannot optimise the edge weights (Mateos *et al.*, 2019).

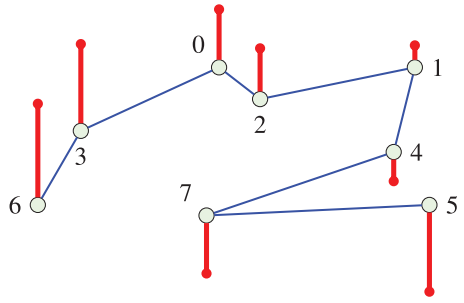
Example 31: Figures 9.1 and 9.2 show that different graph connections can exhibit different smoothness features, given the same observed samples, $\mathbf{x}$, shown in Figure 9.1(a). As also indicated in Figure 9.2, the observed sample retains the lowest frequency components for the graph in Figure 9.1(b) and the highest frequency components for the graph in Figure 9.1(c). This is reflected in a smaller smoothness value, $\mathbf{x}^T\mathbf{L}\mathbf{x}$, for the graph in Figure 9.1(b). This exemplifies that, given the observed graph samples, the smoothness prior is convenient for learning a graph.

9.3 Diffusion Models

It is important to notice that the smoothness that arises from the factor model is imposed in a global manner, which is effective in learning the main structure of a graph. However, promoting the global smoothness can also yield to the overestimation of the details within a graph. To resolve this issue, we can further assume that the observed graph signals are generated via a more complex and powerful model, such as the diffusion model. As shall be discussed in detail in Section 10.6.2, the polynomial filter is a typical choice for treating the diffusion from a

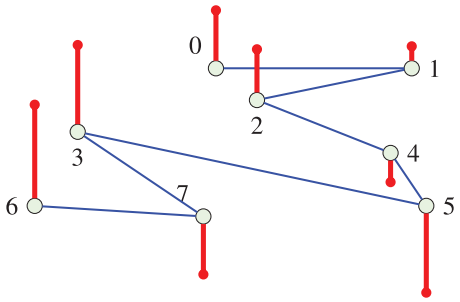


signal $\mathbf{x}$ at the vertices $\{0, 1, 2, 3, 4, 5, 6, 7\}$



signal $\mathbf{x}$ on a path graph with smoothness $\mathbf{x}^T\mathbf{L}\mathbf{x} = 0.94$

(a) (b)



signal $\mathbf{x}$ on a path graph with smoothness $\mathbf{x}^T\mathbf{L}\mathbf{x} = 13.94$ (c)

Figure 9.1: Smoothness and graph learning. (a) The observed graph signal $\mathbf{x} = [0.7, 0.2, 0.6, 1.1 - 0.3, -1.1, 1.3, -0.7]^T$, with (b)–(c) two types of possible path graph connections resulting in different smoothness values, $\mathbf{x}^T\mathbf{L}\mathbf{x}$.

graph data analytics perspective. The benefits arising from learning a graph based on the diffusion model are mainly three-fold:

- Analytical and computational ease during learning.
- The (weak) stationarity is ensured in the generation system (Mateos *et al.*, 2019).
- Ability to control the local smoothness in the model.

The diffusion model is given by (4.28)

$$\mathbf{x} = \sum_{m=0}^{M} h_m \mathbf{S}^m \mathbf{v} + \boldsymbol{\epsilon}, \tag{9.22}$$

where $\mathbf{v}$ is white Gaussian noise $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, while similar to the factor model in (9.19), $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \alpha^2\mathbf{I})$. From (9.22), recall that $\mathbf{S}$ is the (symmetric) shift operator which can be chosen as e.g., the adjacency

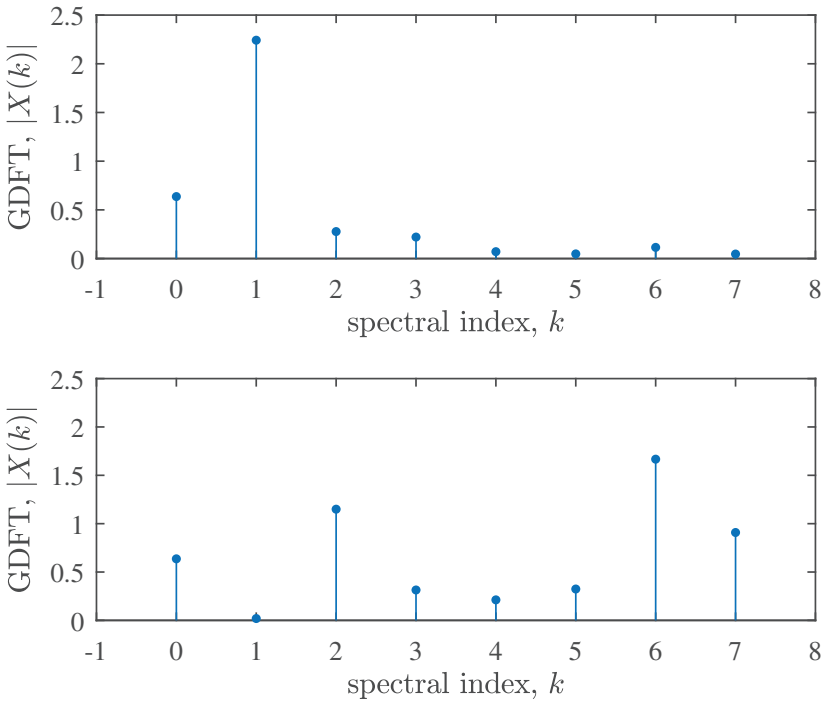


Figure 9.2: Graph signal spectrum values which correspond to the two types of graph connections in Figure 9.1. The top panel corresponds to Figure 9.1(b) and the bottom panel to Figure 9.1(c). The energy is calculated via $\mathbf{x}^T\mathbf{L}\mathbf{x}$, where small values indicate a smooth graph.

matrix, or the Laplacian matrix, to name but a few. Here, we continue to use

$$\mathbf{S} = \mathbf{L}$$

as in Section 4.5. Furthermore, expression (9.22) can be compactly written in the form of (4.28), as

$$\mathbf{x} = \sum_{m=0}^{M} h_m \mathbf{L}^m \mathbf{v}, \tag{9.23}$$

where $\mathbf{L}^0 = \mathbf{I}$ and $h_0 = \alpha^2$ retain the same statistics as those in (9.22).

On the basis of (9.23), the covariance of $\mathbf{x}$ can be calculated as

$$\begin{aligned}\boldsymbol{\Sigma} = \mathrm{E}\{\mathbf{x}\mathbf{x}^T\} &= \Big(\sum_{m=0}^{M} h_m \mathbf{L}^m\Big)\mathrm{E}\{\mathbf{v}\mathbf{v}^T\}\Big(\sum_{m=0}^{M} h_m \mathbf{L}^m\Big)^T \\ &= \sum_{m=0}^{M} h_m \mathbf{L}^m \Big(\sum_{m=0}^{M} h_m \mathbf{L}^m\Big)^T = \mathbf{U}^T\Big(\sum_{m=0}^{M} h_m \boldsymbol{\Lambda}^m\Big)^2 \mathbf{U},\end{aligned} \tag{9.24}$$

where we have used the eigendecomposition $\mathbf{L} = \mathbf{U}^T \mathbf{\Lambda} \mathbf{U}$.

Eigenvector estimation. From (9.24), we can see that the eigenvectors of $\mathbf{L}$ are the same as those of the covariance matrix of $\mathbf{x}$. This means, in a straightforward way, that we can infer the eigenvectors of $\mathbf{L}$ from the empirical covariance of the observed data, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P$.

Eigenvalue estimation. After obtaining the eigenvectors, the remaining task is to estimate the eigenvalues of $\mathbf{L}$. Without any additional constraints, it is obvious that arbitrary values can be chosen as the eigenvalues of $\mathbf{L}$, because we can always find a corresponding set of $h_0, h_1, \ldots, h_M$ that satisfies (9.24). Thus, to achieve a unique solution, we need to employ some prior on the function $f(\cdot)$ (Segarra *et al.*, 2017), to arrive at

$$\min_{\mathbf{L}, \mathbf{\Lambda}} f(\mathbf{L}), \quad \text{subject to } \mathbf{L} = \mathbf{U}^T \mathbf{\Lambda} \mathbf{U}. \tag{9.25}$$

For example, when $f(\mathbf{L}) = \|\mathbf{L}\|_0$, the objective function minimises the number of edges, whereas $f(\mathbf{L}) = \|\mathbf{L}\|_2$ minimises the energy of graph edges. The number of edges can also be minimized using convex relation of $f(\mathbf{L}) = \|\mathbf{L}\|_0$ in the form $f(\mathbf{L}) = \|\mathbf{L}\|_1$, as explained in Part II of this monograph and Section 4.5.

Equation (9.23) assumes that the diffusion process starts from the same initial status, that of white Gaussian noise. An enhanced diffusion model has been proposed in Thanou *et al.* (2017) by assuming that the signals are generated from multiple heat diffusion processes

$$\mathbf{x} = \sum_{m=0}^{M} e^{-h_m \mathbf{L}} \mathbf{v}_m. \tag{9.26}$$

Here, $\mathbf{v}_m$ represents the initial state that can also be optimized, and h_m controls the diffusion time (depth). This means that with a small h_m, the kth column of $e^{-h_m \mathbf{L}}$ is localized at the kth vertex. This model can be solved via a dictionary-learning solver by regarding $[e^{-h_0 \mathbf{L}}, e^{-h_1 \mathbf{L}}, \ldots, e^{-h_M \mathbf{L}}]$ as the dictionary $\mathbf{D}$ and $[\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_M]$ as coefficients $\mathbf{V}$. The objective function can now be formulated as

$$\begin{aligned} \min_{\mathbf{L}, \mathbf{X}, h_m} \quad & \|\mathbf{X} - \mathbf{D}\mathbf{V}\|_F^2 + \text{reg}(\mathbf{V}) + \text{reg}(\mathbf{L}), \\ \text{subject to} \quad & \{h_m\}_{m=0}^{M} \leq 0, \end{aligned}$$

where reg(·) denotes a certain regularization; for more detail, we refer to Thanou *et al.* (2017).

10

Graph Neural Networks

An emerging area which considers graphs in conjunction with neural networks is that of graph neural networks (GNNs). The underpinning idea is to combine the universal approximation property of neural networks and the ability of graphs to capture higher-order information in a physically meaningful way, thus equipping GNNs with enhanced expressive and modelling power. Work in this direction has been facilitated by steadily growing computational power and the ever increasing amount of available data. The beginning of graph neural networks (GNNs) can be traced back to basic network structures (Gori *et al.*, 2005; Micheli, 2009; Scarselli *et al.*, 2008) one decade ago, while recent developments have been centered around graph convolutional networks (GCNs). The GCNs benefit from their intrinsic graph structure, which allows to account for complex implicit coupling among data and information aggregation when processing (or filtering) data at each vertex. This is particularly desirable in deep neural network (DNN) techniques, where the involvement of graphs provides a balance between the "black-box" (but powerful) DNNs and the purely mathematical tools such as manifold optimization and manifold learning. Benefiting from prior information embedded into a graph structure, GCNs are capable of not only handling irregular data

but also of alleviating the "black-box" nature of DNNs, thus helping resolve two major open issues with current DNNs.

Recent literature on GCNs (Wu *et al.*, 2019; Zhou *et al.*, 2018) typically considers the learning aspects, while highlighting two key properties of CNNs: (i) stationarity (via shift invariance of convolution operations) and (ii) compositionality (via downsampling or pooling operations). Taking a sightly different viewpoint, we start from the graph itself and proceed to illuminate that certain types of graphs correspond to major trends in GCNs. We also outline the advantages of treating GCNs in this way, such as the possibility to open avenues for the design of novel types of GCNs.

We introduce GNNs from the perspective of a diffusion process, because of the role of diffusion which underpins signal propagation in graphs. With the ability of graphs to provide intrinsic structures when aggregating information, this allows us to describe recurrent GNNs as a kind of diffusion processes of task-oriented models; all in all, an intuitive way to reveal the underlying mechanisms of GNNs. For example, a standard GNN "feed-forwards" (aggregates) input information layer-by-layer towards the output, calculates deviation from the ground-truth, and then back-propagates to improve the aggregation strategy (weight update). Such information flow (or message passing) is also found in the diffusion process, for example, in temperature transfer heat shown in Example 2. Therefore, the diffusion process can be rephrased as a "language" of neural nets, even for the basic gradient descent updating process. A more complex version is addressed in Section 10.3, in the form of the diffusion process with external sources, which serves to establish a link with the well-know label propagation method. We show that label propagation can be basically regarded as a one-layer GNN, which despite not having weights to be optimized is still powerful enough in semi-supervised learning. This is shown to naturally extend to multiple layers of GNNs, whereby the stacked layers perform message passing (also similar to the diffusion process). We also employ the concept of system on a graph to explain spectral GCNs, while spatial GCNs are shown to admit interpretation as a relaxation of spectral GCNs to the localization in graphs.

10.1 Basic Graph Elements Related to GCNs

The following properties of graphs are helpful in understanding the GCNs (for more detail we refer to Section 2.1 of Part I).

- **Property 1**: When $\mathbf{A}$ is binary, i.e., it represents the connection of vertices (adjacency matrix), the number of walks of a length k, between two vertices m and n, is equivalent to the value of the corresponding element a_{mn} of the kth power of $\mathbf{A}$, that is, of $\mathbf{A}^k$. The number of walks between the vertices m and n, that are of length not higher than k, is given by the corresponding element of $\mathbf{B}_k$, where $\mathbf{B}_k = \mathbf{A} + \mathbf{A}^2 + \cdots + \mathbf{A}^k$. Matrix $\mathbf{B}_k$ gives the k-neighborhood of a vertex, which is a set of vertices that are reachable from this vertex through walks within k steps.

- **Property 2**: For any signal on graph, $\mathbf{x}$, the quadratic form of the Laplacian, $\mathbf{x}^T\mathbf{L}\mathbf{x}$, is of the form

$$\mathbf{x}^T\mathbf{L}\mathbf{x} = \frac{1}{2}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} A_{mn}(x(m) - x(n))^2. \tag{10.1}$$

  This indicates that: (1) the Laplacian matrix, $\mathbf{L} = \mathbf{D} - \mathbf{A}$, is positive semi-definite because $A_{mn}(x(m) - x(n))^2 \geq 0$; (2) the smoothness of graph signal, $\mathbf{x}$, can be quantified via $\mathbf{x}^T\mathbf{L}\mathbf{x}$, which ensures that the quadratic form $\mathbf{x}^T\mathbf{L}\mathbf{x}$ is equivalent to the Dirichlet energy of $\mathbf{x}$, which has been widely used in probabilistic graph models.

Remark 15: The smoothness of a graph signal, $\mathbf{x}$, implies that the signal value would not change much from one vertex to another within the neighborhood of vertex n (assessed by $(x(m) - x(n))^2$). However, signal values are allowed to change significantly when the two vertices are not connected (indicated by zero values of A_{mn}). Therefore, the minimization on $\mathbf{x}^T\mathbf{L}\mathbf{x}$ finds the smoothest signal $\mathbf{x}$ on the graph.

Note that the absolute maximum smoothness (minimum of the smoothness index) is achieved for a signal which is constant over all vertices; such signal is equal to the eigenvector corresponding to the smallest

eigenvalue, $\lambda_0 = 0$, of the graph Laplacian, $\mathbf{L}$ (owing to the Rayleigh quotient). More importantly, this yields $\mathbf{1}^T\mathbf{L}\mathbf{1} = \mathbf{1}^T(\mathbf{D} - \mathbf{A})\mathbf{1} = 0$, that is, the smallest eigenvalue is $\lambda_0 = 0$ with the corresponding normalized eigenvector $\mathbf{x} = \mathbf{u}_0 = \mathbf{1}/\sqrt{N}$, where $\mathbf{1}$ denotes an N-dimensional vector of unities.

Connection to the Laplacian Operator in Function Analysis

One way of understanding the role of the Laplacian matrix in measuring signal smoothness is via its continuous time counterpart – the Laplacian operator in functional analysis. The Laplacian operator over a function $f(\vec{r})$ in the Euclidean space is defined as

$$\text{div}(\text{grad}(f(\vec{r}))) = \nabla(\nabla f(\vec{r})) = \Delta f(\vec{r}),$$

where $\text{grad}(\cdot)$ denotes the gradient operator and $\text{div}(\cdot)$ is the divergence operator. For example, in Cartesian coordinates of two dimensions, $\vec{r} = (x, y)$, we have

$$\Delta f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}. \tag{10.2}$$

Similarly, we can also define the Laplacian operator on the graph, whereby the different (and difficult) aspect is the differential operator. Namely, while as in the discrete signal space, the difference operation is defined as $\nabla f(x) = f(x + 1) - f(x)$, which calculates the difference between $f(x + 1)$ and $f(x)$, the differential on a graph is defined for each edge, that is

$$\nabla f_{mn} = f(m) - f(n).$$

This means that, in general, the differential on a graph allows for a different number of directions at each point (vertex), while for the path graph, the differential ∇f_{mn} naturally simplifies into the standard differential in the Euclidean space.

Example 32: To illustrate the role of the graph Laplacian, consider a graph in Figure 10.1, which is a simplified version of Figure 1(a) of Part I. Its adjacency matrix and the corresponding graph Laplacian

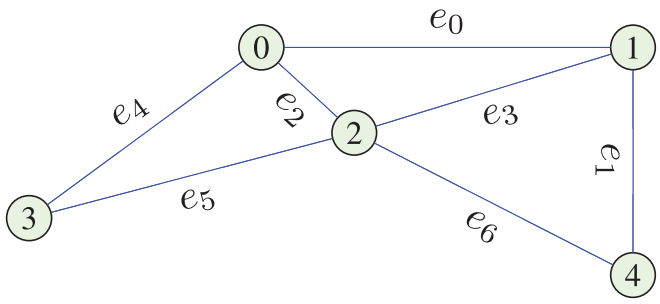


Figure 10.1: A simplified version of the default graph considered throughout this work, as in Figure 1.1(a) in Part I.

matrix are given by

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} 3 & -1 & -1 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 \\ -1 & -1 & 4 & -1 & -1 \\ -1 & 0 & -1 & 2 & 0 \\ 0 & -1 & -1 & 0 & 2 \end{bmatrix}. \tag{10.3}$$

To calculate the gradient, $\text{grad}(\mathbf{f})$, of a signal, $\mathbf{f}$, on this graph

$$\mathbf{f} = \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \\ f(4) \end{bmatrix}, \tag{10.4}$$

which represents the differential at each edge, we introduce the so called incidence matrix, $\mathbf{K}$, given by

$$\mathbf{K} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccccc} e_0 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \end{array}. \tag{10.5}$$

The gradient on the graph now becomes

$$\text{grad}(\mathbf{f}) = \mathbf{K}^T\mathbf{f} = \begin{matrix} \\ e_0 \\ e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \end{matrix} \begin{matrix} \nabla f \\ \begin{bmatrix} f(0) - f(1) \\ f(1) - f(4) \\ f(0) - f(2) \\ f(1) - f(2) \\ f(0) - f(3) \\ f(2) - f(3) \\ f(2) - f(4) \end{bmatrix} \end{matrix}. \tag{10.6}$$

Due to the adjoint property of the divergence operator with regard to inner products, the graph Laplacian for this graph becomes

$$\begin{aligned} \Delta\mathbf{f} &= \text{div}(\text{grad}(\mathbf{f})) = \mathbf{K}(\mathbf{K}^T\mathbf{f}) = (\mathbf{K}\mathbf{K}^T)\mathbf{f} \\ &= \begin{bmatrix} 3 & -1 & -1 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 \\ -1 & -1 & 4 & -1 & -1 \\ -1 & 0 & -1 & 2 & 0 \\ 0 & -1 & -1 & 0 & 2 \end{bmatrix} \begin{bmatrix} f(v_0) \\ f(v_1) \\ f(v_2) \\ f(v_3) \\ f(v_4) \end{bmatrix}. \end{aligned} \tag{10.7}$$

It is now obvious that $\mathbf{K}\mathbf{K}^T$ is equivalent to the graph Laplacian matrix $\mathbf{L}$ in (10.3).

Remark 16: The analysis in (10.4)–(10.7) exemplifies that a graph effectively defines local coordinates with a prior or learnt linkage information, and thus in some sense it can then be considered as a discrete approximation to a manifold. This insight is particularly useful in the design and interpretation of GNNs.

10.2 Gradient Descent as a Diffusion Process

Consider a physical diffusion process, and in particular the Newton's law of cooling, which states that the energy (or heat) loss rate is proportional to the temperature difference between the body (node) and its surrounding environment. The diffusion process can then be understood as an iterative process that converges toward the state of minimum energy, given by $\mathbf{x}^T\mathbf{L}\mathbf{x}$, from any initial condition. Since the

gradient of energy is

$$\text{grad}(\mathbf{x}^T\mathbf{L}\mathbf{x}) = \frac{\partial(\mathbf{x}^T\mathbf{L}\mathbf{x})}{\partial \mathbf{x}^T} = 2\mathbf{L}\mathbf{x},$$

the iterative discrete-time solution for the diffusion process, at an instant $(t+1)$, is given by

$$\mathbf{x}_{t+1} - \mathbf{x}_t = -\alpha \mathbf{L}\mathbf{x}_t, \tag{10.8}$$

or

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \mathbf{L}\mathbf{x}_t,$$

where α is a constant. This solution to the diffusion process can also be formulated as

$$\nabla x(n) \approx -\alpha \sum_{m \in \mathcal{V}_n} (x(n) - x(m)), \tag{10.9}$$

where $\mathcal{V}_n$ is the set of vertices within the neighborhood-one of the vertex n, while $\sum_{m\in\mathcal{V}_n}(x(n)-x(m))$ denotes an aggregate temperature difference between the vertex n and its surrounding vertices.

Remark 17: Equation (10.8) models the change in temperature along time, starting from an initial state $\mathbf{x}_0$. In the following, we will show that this provides an ideal means for designing recurrent GNNs. For more detail on Recurrent Neural Networks (RNN), we refer to Mandic and Chambers (2001) and Mandic and Goh (2009).

The quadratic term, $\mathbf{x}^T\mathbf{L}\mathbf{x}$, is frequently used in data analytics on graphs, for example for estimating smoothness. The gradient of $\mathbf{x}^T\mathbf{L}\mathbf{x}$ is $\partial(\mathbf{x}^T\mathbf{L}\mathbf{x})/\partial\mathbf{x} = 2\mathbf{L}\mathbf{x}$, so that the diffusion process in (10.8) will find the exact minimum of this quadratic form. As mentioned in Section 10.1, the minimum of $\mathbf{x}^T\mathbf{L}\mathbf{x}$ is reached for a constant eigenvector with all elements equal to 1, which indicates that such a diffusion process, when left without any external sources, will eventually settle to the same temperatures for all vertices.

10.3 Label Propagation as a Diffusion Process with External Sources

The stable state (equilibrium) of a diffusion process without external sources cannot give us any useful information because in this case the

data at all the vertices have the same value (i.e., the lowest entropy on the graph). In Physics, we can alter the stable state by adding some constant external sources, which ensures that the final temperatures are not all the same but exhibit some fluctuations governed by their inherent relationships. This is also the basic idea behind many graph machine learning approaches, especially in semi-supervised learning tasks, such as the *label propagation* given in Algorithm 4.

Algorithm 4. Label Propagation

1: **procedure** INITIALIZATION
2: Initialise a graph by treating each data sample separately, as a single vertex;
3: Connect all vertices in the graph, whereby edge weights are defined by some similarity measure;
4: Assign the labels from the labeled samples to the corresponding vertices;
5: Randomly assign values to the unlabeled vertices.
6: **while** Not converged: **do**
7: Propagate from the labeled to the unlabeled vertices: $\mathbf{x} \leftarrow \mathbf{L}\mathbf{x}$. ▷ Diffusion process
8: Re-assign the original labels to the labeled vertices, $\mathbf{x}_L$. ▷ Keep external resources
9: **return** $\mathbf{x}$

The final state of this modified diffusion process can be easily shown to be Zhu (2005),

$$\mathbf{x}_U = (\mathbf{I} - \mathbf{L}_{UU})^{-1}\mathbf{L}_{UL}\mathbf{x}_L, \tag{10.10}$$

where

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{LL} & \mathbf{L}_{LU} \\ \mathbf{L}_{UL} & \mathbf{L}_{UU} \end{bmatrix}, \tag{10.11}$$

and the subscripts U and L designate respectively the unlabelled and labelled sets. Note that for a graph shift, instead of $\mathbf{L}$ we may also use $\mathbf{A}$.

The final stable state will now no longer have the same signal values for all vertices (at least $\mathbf{x}_U \neq \mathbf{x}_L$). This is due to the "external constant" sources of the labelled samples (Line 8 in Algorithm 4), which ensures that the diffusion process results in stable states with signals which are different for each vertex; this also gives the predicted labels for unlabelled signal samples (or vertices) in the inner structures of the graph.

Example 33: To provide a simple illustration of label propagation in handwritten digit recognition, we used three sets of handwritten digits, 1, 5 and 9, each with ten images from the MNIST database (LeCun *et al.*, 1998). We adopted the structural similarity (SSIM) metric (Wang *et al.*, 2004) to measure the similarity between images and to construct a graph accordingly, as shown in Figure 10.2(a). In this example, we chose only two labels for each digit type to act as the external sources in the diffusion process. It needs to be pointed out that without the external sources, the final state would settle to a constant vector; this does not provide any informative predictions.

The total of six given labels are annotated in Figure 10.2(c) and the predicted labels are shown in Figure 10.2(d). By comparing with the ground truth shown in Figure 10.2(b), we can see that label propagation achieved adequate prediction accuracy, given a correctly constructed graph. The level of certainty in the prediction is designated by the node color in Figure 10.2(d), with the provided labels (ground truth) in the red color, and the nodes on the intersections of two types of digits in green colors, indicating the large uncertainty of predictions in these vertices. Therefore, when regarding label propagation as a diffusion process, the temperature can be interpreted as the level of certainty, whereby the external sources (the six given labels) have the highest temperature (designated with the red color) and the heat diffusion performs "certainty propagation". Vertices surrounding the external sources, as a consequence, would retain relatively high temperatures (we are much more sure about the predictions on these nodes).

10.4 GNNs of a Recurrent Style

Now that we have shown that different diffusion models can be utilized to aggregate information across graph vertices, we may employ diffusion to design neural networks on graphs, as neural networks also rely upon information aggregation. One such frequently used recurrent GNN was

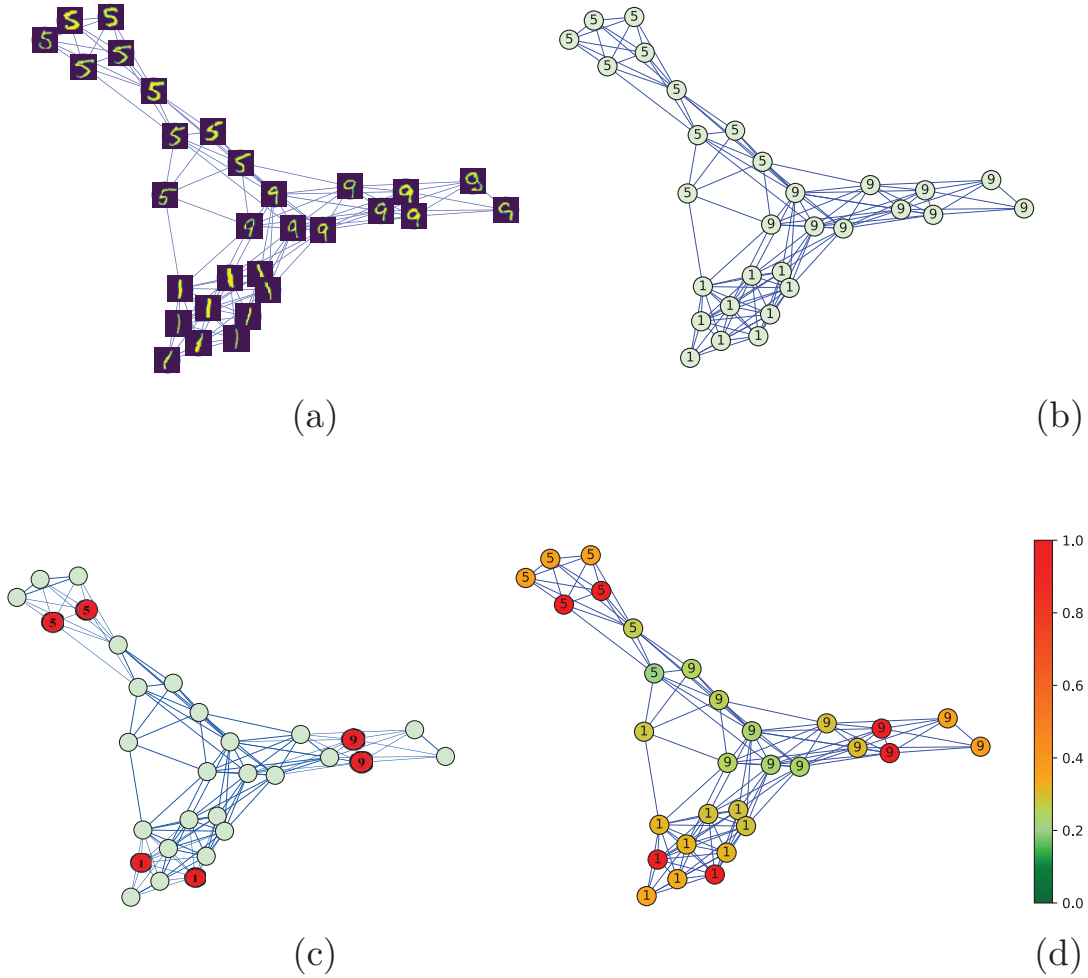


Figure 10.2: Principle of label propagation. We used two labelled images out of ten available images per digit from the MNIST dataset. Three sets of digits (1, 5 and 9) are chosen and each set contains ten images. (a) The resulting graph constructed via the SSIM metric, where two images (nodes) are connected when their SSIM is larger than a threshold (set to 0.35). (b) The ground truth labels for the 30 images considered. (c) Only two labels are provided for each set of images, as indicated by the red color. (d) Predicted labels from the given six (i.e., 2×3) labels via label propagation over the graph Laplacian matrix $\mathbf{L}$. The color bar designates the certainty of predictions, namely, the red color denotes an almost sure prediction with probability approaching 1 and green color poor prediction.

proposed by Scarselli *et al.* (2008), which aggregates information as

$$x_{t+1}(n) = \sum_{m \in \mathcal{V}_n} f(x_t(n), q(n), x_t(m), q(m)), \tag{10.12}$$

$$o(n) = \phi(x(n), q(n)), \tag{10.13}$$

where $x_t(x)$ is the signal value at the nth vertex at a time instant t, $\mathcal{V}_n$ denotes the neighborhood-one of the vertex n, $q(n)$ is a pre-defined feature of $x(n)$, $q(m)$ represents the pre-defined features at the neighbor vertices, and $o(n)$ is the output at the nth vertex. The operators $f(\cdot)$ and $\phi(\cdot)$ can be chosen to form neural networks so that they can be learnt via back-propagation; in other words, the diffusion style model can be learnt from data samples. In a particular case when $q(n)$ and $q(m)$ are omitted, and

$$f(x(n), q(n), x(m), q(m)) = (x(n) - x(m)).$$

Equation (10.12) turns into the original diffusion process in (10.9).

The aggregation function in (10.12) motivates much recent work on GNNs and spatial GCNs, however, this variant of recurrent GNNs needs to undergo the diffusion process until convergence, for every iteration of back-propagation. Moreover, the mapping $f(\cdot)$ in (10.12) needs to be carefully designed to be a contraction mapping to ensure convergence (Mandic, 2007). More recent efforts to improve this model include the gated recurrent GNN (Li *et al.*, 2015) that employs a gated unit as $f(\cdot)$ to ensure convergence within a fixed number of steps, while stochastic steady-state recurrent GNNs (Dai *et al.*, 2018) perform update in (10.12) in a stochastic manner.

Another approach which *incorporates both spatial convolutions and temporal diffusions*, is the diffusion convolution neural network (DCNN) (Atwood and Towsley, 2016), which can be formulated as

$$\mathbf{h}^l = \phi(\mathbf{w}^l \odot \mathbf{L}^l \mathbf{x}), \tag{10.14}$$

where $\mathbf{h}^l$ is the hidden state of the lth layer, $\mathbf{w}^l$ are convolution kernels that are to be learnt, and $\mathbf{L}^l$ is the power series up to l of a certain probability transition matrix (in this case graph Laplacian $\mathbf{L}$) which is similar to Line 7 in Algorithm 4; recall that $\odot$ denotes the element-wise product and ϕ the activation function. It should be pointed out that the model in (10.14) implies that $\mathbf{h}^l$ does not depend on the previous layer (state) $\mathbf{h}^{l-1}$, and that the dimensions of each layer need to be the same; this limits the number of degrees of freedom in the design. The overall output of this GCN is a composition of all layers $\{\mathbf{h}^l\}_{l=1}^{L}$, so

that (10.14) can be interpreted as a set of diffusion processes of different depths (by regarding l as time instant t).

Another way of understanding the operation in (10.14) is that each diffusion step, $\mathbf{L}^l\mathbf{x}$, aggregates (to a certain degree) the heat (or general features and labels). This is a kind of message passing and aggregation that equips the network with the ability to extract statically salient features, which belong to spatial GCNs, introduced below.

Remark 18: Almost all approaches to recurrent GNNs aim to find an efficient and stable diffusion strategy to propagate and aggregate the labels or information at each vertex, so as to facilitate reliable and robust predictions at the final stable stage of the GNNs.

10.5 Spatial GCNs via Localization of Graphs

It is important to note that while CNNs have been an enabling technology for modern machine learning applications, they also suffer from the limitations inherited from the underlying assumption of a regular time/space grid sampling, such as in images and videos. The effort to extend CNNs to GCNs that are able to operate on data acquired on irregular domains therefore needs to accommodate both the convolution (to learn local stationary features) and the pooling (to compose multi-scale patterns) operators. Our main focus is on ways to accommodate the data on irregular domains, while the generalization of pooling is naturally related to the downsampling on the graph (see Part II and Bacciu and Di Sotto, 2019; Ioannidis *et al.*, 2019a; Sakiyama *et al.*, 2019; Tanaka and Eldar, 2019; Zhang *et al.*, 2019a). The key difficulty in defining the convolution on a graph is the absence of a rigorous translation (shift) operator. To this end, the basic idea behind spatial GCNs is the information aggregation principle, which is very similar (sometimes even intertwined with) to the diffusion GNNs in Section 10.4. Instead of waiting for a stable state (along the time instants) of recurrent GNNs, spatial GCNs directly aggregate information by the stacked layers, which is also called message passing. The initial work in this area was by Micheli (2009), the so called neural network for graphs (NN4G). A more general model is the *message passing neural networks*

(MPNNs) (Gilmer *et al.*, 2017), which is given by

$$x^{l+1}(n) = \phi\Big(x^l(n), \sum_{m \in \mathcal{V}_n} f(x^l(n), x^l(m), e_{nm})\Big), \tag{10.15}$$

where $x^l(n)$ represents the data value at the nth vertex of the lth layer, e_{nm} denotes the edge between the nth and the mth vertex, while $f(\cdot)$ is the message passing function and $\phi(\cdot)$ denotes the activation (or vertex updating) function. The model in (10.15) caters for many GCNs, such as those in Micheli (2009) and Kipf and Welling (2016a) which all have different forms of functions $f(\cdot)$ and $\phi(\cdot)$. This model also involves the basic steps for processing graph signals in the spatial domain, i.e., by aggregating the previous messages and passing to the next layer. Bacciu *et al.* further extended this idea to a probabilistic framework Bacciu *et al.* (2018), which enables a probabilistic explanation on each state of each layer.

Furthermore, instead of looking for all neighbors of the central vertex in (10.14), the GraphSAGE approach proposes to sample several neighbors around every vertex (Hamilton *et al.*, 2017), as follows

$$x^{l+1}(n) = \phi(\mathbf{W}^l \cdot \text{concat}\{x^l(n), f\{x^l(m), m \in \widetilde{\mathcal{V}}_n\}\}), \tag{10.16}$$

where concat$\{\cdot,\cdot\}$ denotes the concatenation and $f\{\cdot\}$ the aggregation function, $\mathbf{W}^l$ is the matrix of learnable parameters, and $\widetilde{\mathcal{V}}_n$ denotes a randomly chosen neighbor of the nth vertex. This strategy allows for a mini-batch operation on graphs, which is extremely useful for large graphs.

A further possible improvement is to learn the weights while choosing the neighboring vertices; this includes the graph attention network (GAT) (Veličković *et al.*, 2017), and the mixture model network (MoNet) (Monti *et al.*, 2017). Within GATs, an attention weight, $\alpha_{n,m}$, is added to the parameters in (10.15), which allows us to assign different importance levels to vertices, even within the same neighborhood. The attention weight can be further learnt from an additional convolution sub-network, as proposed in Zhang *et al.* (2018a). On the other hand, the MoNet defines the weights of neighboring edges as a consequence of local coordinates, which has an intrinsic link with the manifolds. More specifically, it defines the importance of the edge connecting the nth

and mth vertex as a probability, p, over some local coordinates, $\mathbf{u}(m,n)$, which reflects the difference (or distance) between the nth and mth vertex. Then, the nth vertex can be aggregated via a specially defined convolution, given by

$$(\mathbf{x} * \mathbf{g})(n) = \sum_{j=1}^{J} g_j \sum_{m \in \mathcal{V}_n} p(\mathbf{u}(m,n))x(m), \tag{10.17}$$

where g_j is the jth index (element) of the convolution kernel, $\mathbf{g}$. In Monti *et al.* (2017), the probability, $p(\mathbf{u}(m,n))$, was chosen as a Gaussian mixture model, which has J clusters to cater for the size of convolution kernel. It has also been shown that the framework of (10.17) accounts for various geometric deep neural networks, through a choice of different local coordinates and weight functions.

10.6 Spectral GCNs via Graph Fourier Transform

As shown in Section 10.5, message passing via the convolution operation plays a crucial role in spatial GCNs. Here, we focus on the methods that operate in a transfer domain and benefit from the mathematically well-defined convolution in the graph spectral domain to yield a class of spectral GCNs.

10.6.1 Graph Fourier Transform

Due to the positive semi-definiteness of $\mathbf{L}$, there are N (the number of vertices) real-valued eigenvalues ($\lambda_0 = 0 \leq \lambda_1 < \lambda_2 < \cdots < \lambda_{N-1}$), which correspond to N distinct orthogonal eigenvectors ($[\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}]$). As mentioned in Section 10.1, the quadratic form, $\mathbf{x}^T\mathbf{L}\mathbf{x}$, measures the smoothness of the data, $\mathbf{x}$, on a graph. Further, when $\mathbf{x}$ represents one of the eigenvectors, $\mathbf{u}_j$, the term $\mathbf{x}^T\mathbf{L}\mathbf{x}$ then measures the smoothness of the eigenvectors, $\mathbf{u}_j^T\mathbf{L}\mathbf{u}_j = \lambda_j$. The matrix of eigenvectors, $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_N]$, represents an orthogonal transform basis, which is similar to principal component analysis (PCA), while benefiting from a physically more important and beneficial property in practice because the graph Laplacian bases indicate the smoothness of eigenvectors.

Remark 19: Through multiplication of the data, $\mathbf{x}$, by the eigenmatrix, $\mathbf{U}\mathbf{x}$, the original data $\mathbf{x}$ are decomposed into different constituent components, which vary from the most smooth to the most non-smooth. This is exactly the principle of the Fourier transform, which transforms a signal to different frequency components (bases). In this case, λ_j has the physical meaning of (squared) frequency, as shown in Section 3.5.2 of Part II. In particular, when the graph structure is a path graph, the original Fourier transform is obtained.

Based on the graph Fourier transform, covered in detail in Part II of this monograph, we can now define the graph convolution operator which states that the convolution in the spatial (vertex) domain is equal to the multiplication in the spectral domain. This bypasses the requirement for translation (or shift operator) to define convolution in the vertex domain, whilst maintaining the concept of "convolution" over graph signals. In this way, the graph convolution is given by

$$\mathbf{U}^T(\mathbf{x} * \mathbf{g}) = (\mathbf{U}^T\mathbf{x}) \odot (\mathbf{U}^T\mathbf{g}), \tag{10.18}$$

where $\mathbf{x}$ and $\mathbf{g}$ are two vectors whose elements are the data values at vertices $n \in \mathcal{V}$. Recall that $\mathbf{U}$ in (10.18) is the Fourier basis composed by the eigenvectors of $\mathbf{L}$ and $\odot$ denotes the Hadamard (element-wise) product. It is worth mentioning that the matrix $\mathbf{U}$ is a graph counterpart of the frequency shift operator in the continuous time Fourier transform.

10.6.2 Graph Spectral Filtering as Multiple Diffusion Processes

Upon inspection of the diffusion process of the cooling law in Section 10.3, we can see that it actually aggregates the data values at the connected vertices to process the current vertex. Consider now a polynomial filter of the diffusion process, given by

$$\mathbf{x} \leftarrow \mathbf{B}_k\mathbf{x} = (\mathbf{A} + \mathbf{A}^2 + \cdots + \mathbf{A}^k)\mathbf{x}, \tag{10.19}$$

where k neighboring vertex data values are aggregated to produce the current vertex data sample, according to the Property 1 of Section 10.1. It can be proved that the k-neighboring property also holds when $\mathbf{B}_k$ is given by the powers of the Laplacian, $\mathbf{L}^k$ (Lemma 5.4, Hammond *et al.*, 2011), as we are still using the k-neighbor information when aggregating,

that is

$$\mathbf{x} \leftarrow (\mathbf{L} + \mathbf{L}^2 + \cdots + \mathbf{L}^k)\mathbf{x}. \tag{10.20}$$

Upon rewriting (10.20) in the graph spectral domain, we have

$$\mathbf{x} \leftarrow \mathbf{U}(\mathbf{\Lambda} + \mathbf{\Lambda}^2 + \cdots + \mathbf{\Lambda}^k)\mathbf{U}^T\mathbf{x}, \tag{10.21}$$

or equivalently

$$\mathbf{X} \leftarrow (\mathbf{\Lambda} + \mathbf{\Lambda}^2 + \cdots + \mathbf{\Lambda}^k)\mathbf{X}, \tag{10.22}$$

where $\mathbf{X}$ is the spectral representation of $\mathbf{x}$, through $\mathbf{X} = \mathbf{U}^T\mathbf{x}$, and $\mathbf{\Lambda}$ a diagonal matrix of which the elements are the ordered eigenvalues of $\mathbf{L}$. By combining (10.18) and (10.22), the convolution operation on the graph can be chosen as

$$\mathbf{U}^T\mathbf{g} = \text{poly}(\mathbf{\Lambda}) = \mathbf{\Lambda} + \mathbf{\Lambda}^2 + \cdots + \mathbf{\Lambda}^k. \tag{10.23}$$

We should point out that although there are many choices for the convolutional filter, $\mathbf{g}$, we typically choose the polynomial kernel as $\text{poly}(\mathbf{\Lambda}) = \mathbf{\Lambda} + \mathbf{\Lambda}^2 + \cdots + \mathbf{\Lambda}^k$, which ensures the localization in the vertex domain within k-neighbors.

10.6.3 Graph Spectral Filtering via Neural Networks

Given the importance of convolution in the modelling of data propagation on graphs, and the computational difficulties in its evaluation, it is natural to employ neural networks to implement the function $\mathbf{g}$ in (10.23), per layer. In this way we also take advantages of the spatial convolution operations and the universal approximation property of neural networks. This forms the basis of various spectral GCN methods.

The spectral GCN was proposed by Bruna *et al.* (2013), and is based on a simple spectral model given by

$$\mathbf{x}_j^{l+1} = \phi\left(\mathbf{U}\sum_{i=1}^{c_l}\mathbf{\Theta}_{i,j}^l\mathbf{U}^T\mathbf{x}_i^l\right) \quad j = 1, 2, \ldots, c_{l+1}, \tag{10.24}$$

where l represents the index of each layer, c_l is the number of filters (channels) of the lth layer, $\mathbf{\Theta}_{i,j}^l$ is a diagonal matrix which contains the set of learnt parameters of the lth layer, and $\phi(\cdot)$ is the activation function of neurons. In (10.24), the summation ensures the aggregation

of features filtered by different convolutional kernels, $\mathbf{\Theta}_{i,j}^{l}$, which is similar to a linear combination across kernels in CNNs. Although it achieves graph convolution through neural networks, this methodology has two main limitations: (i) the localization in the vertex domain cannot be ensured by $\mathbf{\Theta}_{i,j}^{l}$, although it is a key to the success of convolutional neural networks in extracting local stationary features; (ii) computational burden arising from the $\mathcal{O}(N^2)$ multiplications of $\mathbf{U}$ and $\mathbf{U}^T$, and the eigendecomposition of $\mathbf{L}$ to obtain $\mathbf{U}$ may be prohibitive for large graphs.

A possible way of mitigating these issues is to employ a polynomial form similar to that of (10.22), as mentioned in Section 10.6.2. This both relieves the first issue of the localization, and helps to control a balance between the localization in the vertex domain and the localization in the spectral domain (see Part II of this monograph). More specifically, to further improve the localization in the spatial domain in order to extract local patterns, we promote smoothness in the spectral domain through filtering by poly($\mathbf{\Lambda}$), whereby the term poly($\mathbf{\Lambda}$) is designed with a set of learnable parameters $\mathbf{\Theta} = \{\theta_i\}_{i=1}^{k}$, in the form

$$\text{poly}_{\mathbf{\Theta}}(\mathbf{\Lambda}) = \theta_1\mathbf{\Lambda} + \theta_2\mathbf{\Lambda}^2 + \cdots + \theta_k\mathbf{\Lambda}^k. \tag{10.25}$$

In this way, the update rule of (10.21) can now be rewritten as

$$\mathbf{x} \leftarrow \text{poly}_{\mathbf{\Theta}}(\mathbf{L})\mathbf{x} = \mathbf{U}\text{poly}_{\mathbf{\Theta}}(\mathbf{\Lambda})\mathbf{U}^T\mathbf{x}. \tag{10.26}$$

Notice that in (10.26), the multiplication by $\mathbf{U}$ is not necessary at every layer, but the powers of $\mathbf{L}$ are needed and are computational demanding. On the basis of (10.26), Defferrard *et al.* (2016) further proposed the Chebyshev graph neural network, which employs the Chebyshev polynomial to ease the computation burden of poly$_{\mathbf{\Theta}}(\mathbf{\Lambda})$, in the form

$$\text{poly}_{\mathbf{\Theta}}(\mathbf{\Lambda}) = \sum_{i=1}^{k} \theta_i T_i(\widetilde{\mathbf{\Lambda}}), \tag{10.27}$$

where $\widetilde{\mathbf{\Lambda}} = {}^{2\mathbf{\Lambda}}/_{\lambda_{\max}} - \mathbf{I}_N$, while $T_i(\widetilde{\mathbf{\Lambda}})$ is the Chebyshev polynomial that has an easy-to-compute recurrent form $T_i(\widetilde{\mathbf{\Lambda}}) = 2\widetilde{\mathbf{\Lambda}}T_{i-1}(\widetilde{\mathbf{\Lambda}}) - T_{i-2}(\widetilde{\mathbf{\Lambda}})$ ($T_0(\widetilde{\mathbf{\Lambda}}) = \mathbf{I}$, and $T_1(\widetilde{\mathbf{\Lambda}}) = \widetilde{\mathbf{\Lambda}}$). With this Chebyshev polynomial, we are

able to elegantly avoid the computation of the powers of $\mathbf{L}$, through

$$\mathbf{x} \leftarrow \text{poly}_{\boldsymbol{\Theta}}(\mathbf{L})\mathbf{x} = \sum_{i=1}^{k} \theta_i T_i(\widetilde{\mathbf{L}})\mathbf{x}, \tag{10.28}$$

where $\widetilde{\mathbf{L}} = {}^{2\mathbf{L}}/_{\lambda_{\max}} - \mathbf{I}_N$. This framework significantly reduces the computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(kN)$, and has been widely used in various graph learning tasks. Recent work Kipf and Welling (2016a) further simplifies (10.28) by only employing the first-order Chebyshev polynomial ($k = 1$), which achieves superior performances in semi-supervised learning. The authors claimed that it is unnecessary to employ a k-order format because the first-order Chebyshev polynomial is sufficient to mitigate overfitting, while the localization of k-neighbors can be achieved by stacking layers of neural networks.

Despite mathematical elegance and physical intuition, spectral GCNs have been mainly limited to fixed network structures during both training and testing. More specifically, when employing spectral GCNs, the graph connections should be ascertained in advance because even a slight change in a graph connection would lead to a totally different eigenbasis. This, in turn, means that the whole graph needs to be initialized before training, which implies that spectral GCNs cannot be trained in a mini-batch manner, as the trained model is domain dependent.

Example 34: To illustrate an implementation of one typical spectral GCN (Kipf and Welling, 2016a) in semi-supervised learning, we employed the Cora dataset (Motl and Schulte, 2015) that contains 2708 machine learning related publications with seven classes (case based, genetic algorithms, neural networks, probabilistic methods, reinforcement learning, rule learning and theory). Each publication has a feature vector that indicates whether an article includes any unique selected keywords. Furthermore, the graph is constructed via its citation relationships.

For the GCN method, we employed a Pytorch implementation of the work in Kipf and Welling (2016a) which is available at https://github.com/tkipf/pygcn. The basic structure of the GCN network is illustrated in Figure 10.3, and its pseudo-code is provided in Algorithm 5. In this example, the number of hidden units was set to 256. We used

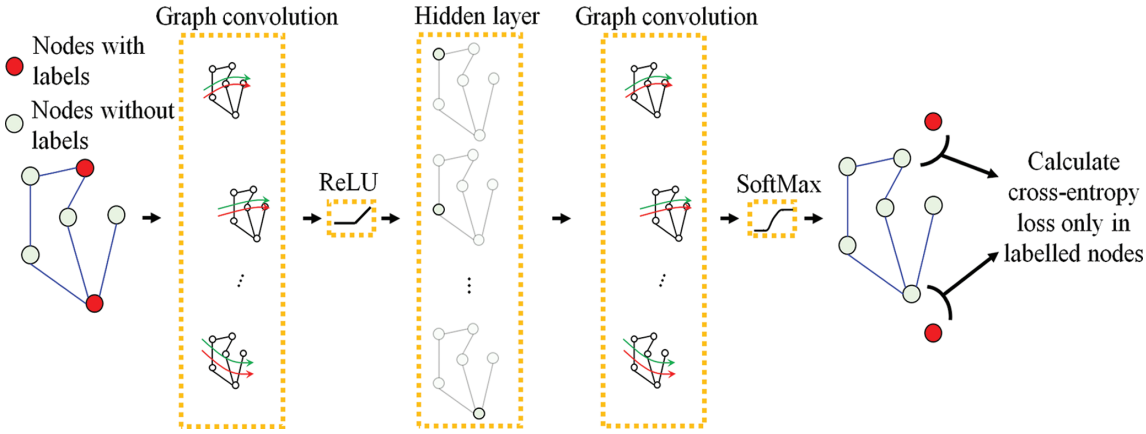


Figure 10.3: The structure of the GCN proposed in Kipf and Welling (2016a) for semi-supervised learning.

Algorithm 5. Training Process of a Typical GCN (Kipf and Welling, 2016a)

Input: Node features, $\mathbf{X}^0 \in \mathbb{R}^{N \times P}$, adjacency matrix, $\mathbf{A} \in \mathbb{R}^{N \times N}$;
while Not converged: **do**
    Layer 1: $\mathbf{X}^1 = \text{GCO}^1(\mathbf{X}^0, \mathbf{A})$
    Output: $\mathbf{X}^2 = \text{GCO}^2(\mathbf{X}^1, \mathbf{A})$
    Loss calculation on $\mathbf{X}^2$ and back-propagation for optimization ▷ One iteration of GCN training
Output: The GCN with optimal $\mathbf{W}$ for each layer
procedure $\text{GCO}^l(\mathbf{X}, \mathbf{A})$ ▷ Graph convolution operation for the lth layer
    Renormalization trick: $\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ ▷ First-order of Chebyshev polynomials of (10.28)
    Graph convolution: $\mathbf{Y} = \tilde{\mathbf{A}} \cdot \mathbf{X} \cdot \mathbf{W}^l$ ▷ $\mathbf{W}^l$ are learnable parameters in the layer
    Non-linearity: $\mathbf{Z} = \text{act}(\mathbf{Y})$ ▷ Examples of activation functions act are sigmoid and ReLU functions
    Return: $\mathbf{Z}$

different ratios of data for training and plotted the test accuracy in classifying those publications into the seven classes in Figure 10.4. Observe that with only 10% of the available samples, a simple GCN with one hidden layer can achieve >80% classification accuracy. It is

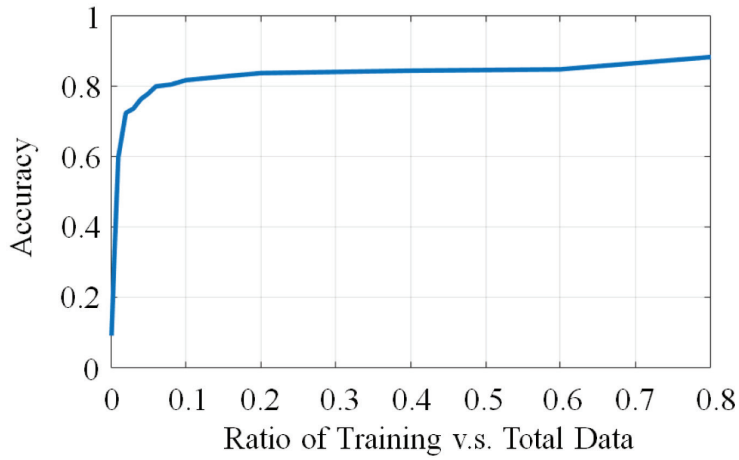


Figure 10.4: Portions of data used for training versus the test accuracy on Cora dataset (Motl and Schulte, 2015). We considered a simple implementation of one typical GCN (Kipf and Welling, 2016a) for semi-supervised learning. In this example, we used one hidden layer with 256 neurons. The dropout rate was set to 0.5 and learning rate to 0.01.

possible to further improve the test accuracy by extending the number of hidden units or increasing network depth. This simple example, however, highlights the powerful learning ability of GCNs on structured data.

10.7 Link Prediction via Graph Neural Nets

Oftentimes, the dynamics of the underpinning problem at hand dictate that it is necessary to establish additional connections in a graph, in addition to the already existing edges. This is achieved through so called *link prediction*, which can be used in both graph completion (interpolation) and graph extension (expansion) (Liben-Nowell and Kleinberg, 2007). A direct way to perform link prediction would be to apply some heuristic similarity method to the vertices and sub-graphs, in order to estimate missing links between the vertices, as is the case with the PageRank method introduced in Section 6.5, SimRank (Jeh and Widom, 2002) and SEAL (Zhang and Chen, 2018) approaches. Alternatively, learning strategies may be employed to infer such links automatically in some "well-behaved" embedded spaces of graphs; this is highly related to the field of graph representation learning, that is, learning a representative latent space given an existing graph.

Remark 20: The spectrum of a graph, elaborated in detail in Part I and Part II of this monograph, is a simple yet effective candidate for an embedding space, since it reflects the smoothness (frequency) of data on a graph. In this way, spectral clusters can be utilized to train a classifier to predict links (Tang and Liu, 2011).

More advanced latent space methodologies for link prediction include Deepwalk (Perozzi *et al.*, 2014), Note2vec (Grover and Leskovec, 2016) and Line (Tang *et al.*, 2015), all of which learn meaningful and continuous low-dimensional latent spaces by preserving (encoding) neighboring information at the vertices. For more detail, we refer to the recent reviews in Wu *et al.* (2019), Zhang *et al.* (2018b), and Chami *et al.* (2020).

More recently, owing to their ability to represent probabilistic generative models, GCNs have also been applied to link prediction tasks, owing to their ability to implicitly process local information in graphs. Within GCNs, two types of generative models are commonly used, the graph variational auto-encoder (VAE) and the graph generative adversarial model (GAN) (Bojchevski *et al.*, 2018; De Cao and Kipf, 2018; Wang *et al.*, 2017). Standard autoregressive models have also been considered to progressively generate graphs (Li *et al.*, 2018; You *et al.*, 2018). We here focus on VAE-based methods because they are designed to straightforwardly learn representations for link prediction, while GAN related methods are motivated by graph generation. We should also point out that the VAE- and GAN-based approaches are not independent, as VAE-based approaches take advantage of additional adversarial modules to enhance learning capacity (Pan *et al.*, 2018; Yu *et al.*, 2018).

The graph VAE (Kipf and Welling, 2016b) employs the VAE framework proposed in Kingma and Welling (2013), with the underpinning idea similar to the probabilistic models covered in Section 9, whereby a signal, $\mathbf{x}$ (or the graph in the case of graph VAEs), is generated by Gaussian random samples, $\mathbf{v}$. However, different from the linear model approaches (Section 9), the VAE employs a neural net (decoder) as a non-linear way of graph generation. To avoid trivial generation from

random noise governed by the distribution $p(\mathbf{x} \mid \mathbf{v})$, we need to find a reasonable set of random samples, $\mathbf{v}$, that is likely to generate meaningful graph signals, $\mathbf{x}$. This resembles an encoder structure, governed by the distribution $q(\mathbf{v} \mid \mathbf{x})$, which can be trained by minimizing the distance between $q(\mathbf{v} \mid \mathbf{x})$ to the true posterior[1] $p(\mathbf{v} \mid \mathbf{x})$. We refer to Doersch (2016) for a detailed tutorial on the VAE. By using the Kullback-Leibler (KL) divergence as the distance metric in the minimization, the VAE arrives at the following relationship

$$\log p(\mathbf{x}) - \mathrm{KL}(q(\mathbf{v} \mid \mathbf{x}) \| p(\mathbf{v} \mid \mathbf{x})) = \mathbb{E}_{q(\mathbf{v} \mid \mathbf{x})}[\log p(\mathbf{x} \mid \mathbf{v})] - \mathrm{KL}(q(\mathbf{v} \mid \mathbf{x}) \| p(\mathbf{v})). \tag{10.29}$$

Observe that by maximizing the right-hand side of (10.29), we are effectively maximizing the log-likelihood of $p(\mathbf{x})$, whilst at the same time minimizing the distance between the true posterior, $p(\mathbf{v} \mid \mathbf{x})$, and the assumed one, $q(\mathbf{v} \mid \mathbf{x})$.

Remark 21: The right-hand side of (10.29) is called the evidence lower bound (ELBO) in Bayesian variational inference; more importantly, it is tractable and yields clear and physically meaningful structures, whereby $q(\mathbf{v} \mid \mathbf{x})$ is an encoder and $p(\mathbf{x} \mid \mathbf{v})$ the corresponding decoder, with $\mathbb{E}_{q(\mathbf{v} \mid \mathbf{x})}[\log p(\mathbf{x} \mid \mathbf{v})]$ as the reconstruction loss, while $\mathrm{KL}(q(\mathbf{v} \mid \mathbf{x}) \| p(\mathbf{v}))$ regularises the feasible set of $\mathbf{v}$ on some well-behaved manifolds in the latent space.

A direct extension of the framework in (10.29) to link prediction would be to estimate the missing graph connections using the VAE. The initial attempt, called the variational graph auto-encoder (VGAE) (Kipf and Welling, 2016b), models the connectivity (adjacency matrix) through the encoder, by $q(\mathbf{v} \mid \mathbf{A}, \mathbf{x})$, whilst the decoder remains the standard neural network used to reconstruct the connectivity. In this way, the encoder in the VGAE can be implemented as a GCN, while the decoder may be simplified to only a product operation $p(\mathbf{A} \mid \mathbf{v}) = \mathrm{sigmoid}(\mathbf{v}\mathbf{v}^T)$, where the sigmoid function is used to satisfy the probability constraint.

[1] The reason we do not directly use $p(\mathbf{v} \mid \mathbf{x})$ is due to the fact that the posterior is intractable when modelling the likelihood $p(\mathbf{x} \mid \mathbf{v})$ through neural nets. Thus, the variational method employs another distribution, $q(\mathbf{v} \mid \mathbf{x})$, and minimises its distance to $p(\mathbf{v} \mid \mathbf{x})$ by means of some tractable formats (i.e., the evidence lower bound, or ELBO for short).

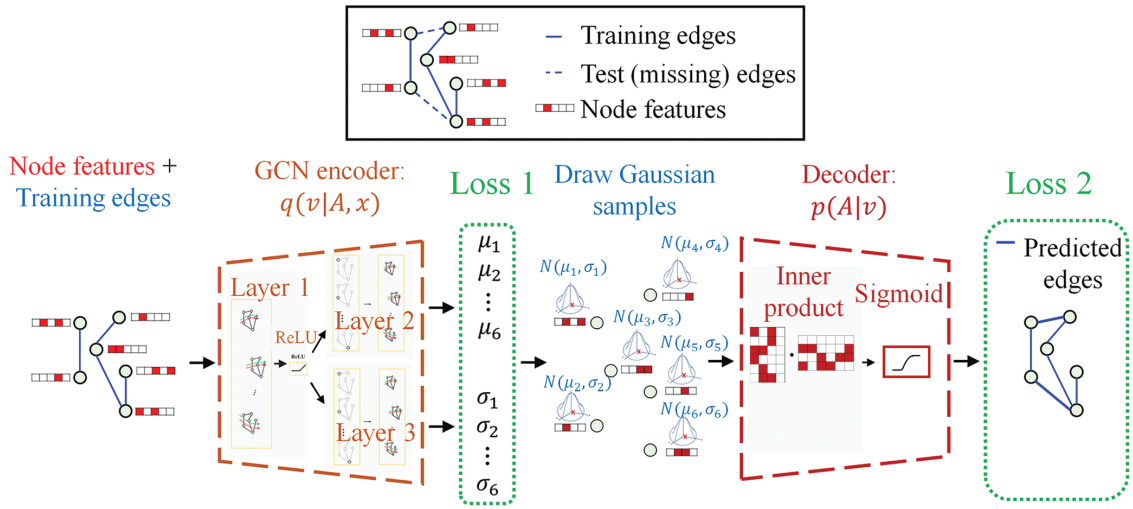


Figure 10.5: Principle of the use of VGAE in link prediction. The training process of the VGAE is based on the provided training edges and node features, whilst the test edges are not connected. Different from the standard auto-encoder, the encoder within the VGAE yields the estimated mean and covariance values of Gaussian distributions ($\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_6$ and $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots, \boldsymbol{\sigma}_6$), while the input to the decoder is randomly drawn from the corresponding Gaussian distributions. Being a graph variance of (10.29), the loss consists of two parts: The term $\mathrm{KL}(q(\mathbf{v} \mid \mathbf{A}, \mathbf{x}) \| p(\mathbf{v}))$ of *Loss 1* which reflects how well the output Gaussian approaches the standard Gaussian distribution, $p(\mathbf{v})$ (zero mean and identity covariance), while the term $\mathbb{E}_{q(\mathbf{v}|\mathbf{A},\mathbf{x})}[\log p(\mathbf{A} \mid \mathbf{v})]$ of *Loss 2* corresponds to the quality of reconstruction of the adjacency matrix, $\mathbf{A}$.

Figure 10.5 provides a closer insight into the VGAE, with the corresponding pseudo-code given in Algorithm 6. To enable the use of a GCN also in the decoder and to comply with graph theory, work in Grover *et al.* (2019) proposes to employ an intermediate and learnable adjacency matrix within the decoder. Other improvements include regularizing VGAE through semantic validity (Ma *et al.*, 2018), use of rich models (such as mixture models) as a prior, $p(\mathbf{v})$ (Hasanzadeh *et al.*, 2019), and an asynchronous message passing scheme for directed acyclic graphs (Zhang *et al.*, 2019b).

Remark 22: The VGAE differs from the standard graph auto-encoder (GAE) in the latent space, which arises from the use of variational inference, in that the representation (embedding) of each graph node is a Gaussian distribution described by the learnt mean and covariance.

A potential problem with the standard GAE is that its reconstruction loss enforces the training of the GAE towards recovering an incomplete training adjacency matrix, whereby the test and validation edges are masked out. Therefore, if a perfect reconstruction was achieved in training, in the test stage, the GAE would only recover the training edges, whilst ignoring the test and validation edges. The way of relieving this issue by the VGAE rests upon its usage of noise and uncertainty inferred through the learning process. However, this also adds disturbance and poses difficulties to the operation of the decoder during reconstruction, as this increases the reconstruction loss in an implicit adversarial way, but forces the VGAE to learn robust and meaningful graph representations.

Most recently, a framework named RCF-GAN has been proposed to seamlessly combine the benefits of the auto-encoder and adversarial learning; this is achieved via a reciprocal requirement in the latent space, while to enhance robustness, characteristic functions are employed in design of the losses (Li *et al.*, 2020).

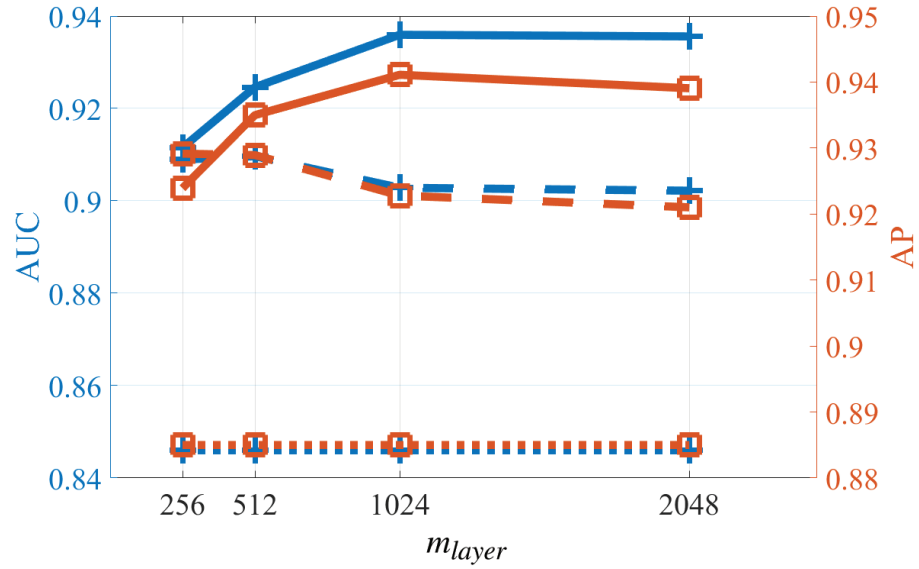
Remark 23: By virtue of the RCF-GAN, a meaningful representation can be learnt in the embedded space; this means that the links can be directly predicted by the embedded features, instead of being reconstructed by the decoder as with the VGAE.

The application of the RCF-GAN in graph link prediction differs from the VGAE in two main aspects: (1) the decoder no longer reconstructs the adjacency matrix but yields the node features, in order to satisfy the reciprocal requirement of the RCF-GAN; (2) the reconstruction loss in the RCF-GAN is based on the node features, and operates directly in the embedded space, as opposed to that based on the adjacency matrix in the data domain of the VGAE. The so learnt embedded features are therefore immediately graph representations and hence generalise well, thus equipping the link prediction via the RCF-GAN with the ability to effectively avoid overfitting of the training edges.

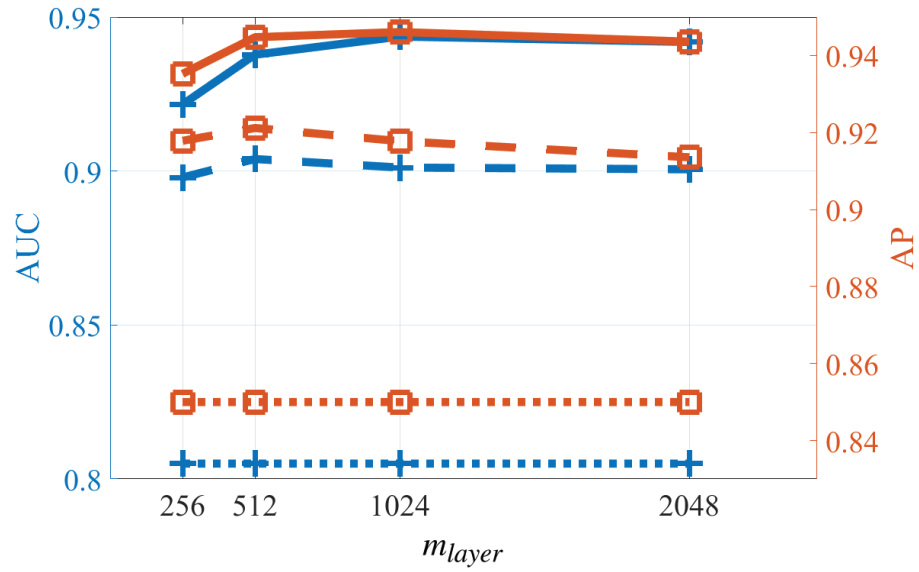
The following example demonstrates that although the test connections were not provided in the training stage, the VGAE and RCF-GAN were still able to successfully recover the missing links through the inner product (decoder) of the trained graph representations.

Example 35: We employed two well-known datasets, Cora and Citeseer (Sen *et al.*, 2008) on a graph link prediction task. For a fair comparison, the split of datasets into training, validation and test sets was exactly the same as that in the VGAE (Kipf and Welling, 2016b), that is, 85% for training, 5% for validation and 10% for testing. For the encoder of the RCF-GAN, we adopted the same basic GCN as that adopted in the VGAE. The dimensions of the encoder for both the RCF-GAN and the VGAE were set to $\{m_{node}, m_{layer}, 128\}$, where m_{node} denotes the dimension of node features and m_{layer} represents the dimension of the middle layer. In the experiments, m_{layer} assumed the values from $\{256, 512, 1024, 2048\}$. The VGAE was also run with different m_{layer}, for a fair comparison with the available VGAE implementation (Kipf and Welling, 2016b). For the generator of the RCF-GAN, we used a simple 3-layer fully connected neural net with dimensions $\{128, m_{layer}, m_{node}\}$. We repeatedly ran the training and testing process 10 times, with both models trained over 300 epochs. Performance was evaluated through the mean values of the area under the ROC curve (AUC) and average precision (AP) metrics, with the results given in Figure 10.6.

Observe from Figure 10.6 that for all m_{layer} the RCF-GAN in link prediction consistently outperformed the VGAE and the spectral clustering methods, with a significant margin. More specifically, the VGAE achieved its best performance at approximately $m_{layer} = 512$, with the obtained AUC (AP) of 0.910 (0.929) for Cora and 0.904 (0.921) for Citeseer. Further increasing m_{layer} may lead to overfitting of the VGAE, thus decreasing its performances in link prediction. However, with the increase in network sizes, the performances of the RCF-GAN improved correspondingly, and the AUC (AP) scores reached 0.936 (0.941) for Cora and 0.944 (0.946) for Citeseer. This may be due to the fact that the RCF-GAN does not directly perform the reconstruction of the adjacency matrix but learns a semantic embedded space for link prediction. This improvement also validates the effectiveness and efficiency of the learnt embedded space within the RCF-GAN.



(a) Cora



(b) Citeseer

Figure 10.6: The AUC and AP scores for graph link prediction using the RCF-GAN and VGAE, with m_{layer} ranging from 256 to 2048. The solid lines show the AUC and AP metrics of the RCF-GAN in testing, whereas the dashed lines designate the results for the standard VGAE. Since spectral clustering based graph link prediction (Tang and Liu, 2011) is a typical baseline for using GCNs, we also plot the spectral clustering results from Kipf and Welling (2016b) in dotted lines.

Algorithm 6. Training Process of VGAE Kipf and Welling (2016b)

Input: Node features, $\mathbf{X}^0 \in \mathbb{R}^{N\times P}$, (incomplete) training adjacency matrix, $\mathbf{A} \in \mathbb{R}^{N\times N}$;
while Not converged: **do**
    Forward process of the encoder:
    Layer 1: $\mathbf{X}^1 = \mathrm{GCO}^1(\mathbf{X}^0, \mathbf{A})$
    Layer 2: $\mathbf{U} = \mathrm{GCO}^2(\mathbf{X}^1, \mathbf{A})$ ▷ Output Gaussian mean vector of each node
    Layer 3: $\mathbf{C} = \mathrm{GCO}^3(\mathbf{X}^1, \mathbf{A})$ ▷ Output Gaussian diagonal covariance vector of each node
    Forward process of the decoder:
    For each node, n, draw one sample, $\mathbf{v}_n$, from $\mathcal{N}(\mathbf{u}_n, \mathbf{c}_n)$, where $\mathbf{u}_n$ and $\mathbf{v}_n$ are the nth rows of $\mathbf{U}$ and $\mathbf{C}$
    Inner product operation: $\hat{\mathbf{A}} = \mathrm{sigmoid}(\mathbf{V}\mathbf{V}^T)$, where $\mathbf{v}_n$ is the nth row of $\mathbf{V}$
    Loss calculation: $\frac{1}{N}\sum_{n=1}^{N} \mathrm{KL}(\mathcal{N}(\mathbf{u}_n, \mathbf{c}_n)\|\mathcal{N}(\mathbf{0}, \mathbf{I})) + \mathrm{BCE}(\hat{\mathbf{A}}, \mathbf{A})$ ▷ BCE denotes the binary cross entropy loss
    Back-propagation for optimization ▷ One iteration of training the VGAE
Output: Optimal embedding $\mathbf{U}$ of nodes
procedure $\mathrm{GCO}^l(\mathbf{X}, \mathbf{A})$ ▷ Graph convolution operation for the lth layer
    Renormalization trick: $\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ ▷ First-order of Chebyshev polynomials of (10.28)
    Graph convolution: $\mathbf{Y} = \tilde{\mathbf{A}} \cdot \mathbf{X} \cdot \mathbf{W}^l$ ▷ $\mathbf{W}^l$ is learnable parameters in the layer
    Non-linearity: $\mathbf{Z} = \mathrm{act}(\mathbf{Y})$ ▷ Examples of activation functions act can be sigmoid and ReLU functions
    Return: $\mathbf{Z}$

11

Tensor Representation of Lattice-Structured Graphs

It is often desirable to generalize graphs in order to account directly for higher-order and higher-dimensional relationships between data sources (Cooper and Dutle, 2012; Saito *et al.*, 2018; Zhou *et al.*, 2007). One such way is via the hypergraph approach, which allows the edges to link more than two vertices (Berge, 1984). Another possibility is through a multi-layer network of graphs, whereby graph vertices reside on a high dimensional regular lattice structure which results from the Cartesian product of several one-dimensional path graphs (for more detail, see Part I). We next show that tensors (multidimensional data arrays) are perfectly suited to model the latter approach. It is further shown that tensors can be considered as a special class of graph signals, which in turn allows the associated adjacency matrices to exhibit a physically meaningful structured form, referred to as *Kronecker summable*. By virtue of the underlying multilinear tensor algebra, this effectively reduces the number of parameters required to model the entire graph connectivity structure (Bacciu and Mandic, 2020).

11.1 Tensorization of Graph Signals in High-Dimensional Spaces

A tensor of *order* M is an M-way data array, denoted by $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_M}$. For example, a vector $\mathbf{x} \in \mathbb{R}^I$ is an order-1 tensor, a matrix $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$ is an order-2 tensor, while a 3-way array $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is an order-3 tensor. The mth dimension of an order-M tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_M}$, is referred to as the mth *mode* which is of size I_m entries.

To establish a relationship between graph signals and tensors, we begin by considering an N-vertex graph, denoted by $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$. With each vertex on the graph we can associate a variable (signal), denoted by $x(n) \in \mathbb{R}$, which maps a vertex, $n \in \mathcal{V}$, to a real, that is, x: $\mathcal{V} \mapsto \mathbb{R}$. In other words, each vertex represents a scalar-valued field in a single-dimensional coordinate system. When considering all N vertices in $\mathcal{V}$, we can form the vector $\mathbf{x} \in \mathbb{R}^N$ which defines the mapping $\mathbf{x}$: $\mathcal{V} \mapsto \mathbb{R}^N$.

On the other hand, if a graph resides in an M-dimensional space, then each vertex, $n \in \mathcal{V}$, has a one-to-one correspondence with a unique coordinate vector in this space, denoted by $(i_1, \ldots, i_M) \in \mathbb{N}^M$, where $i_m \in \mathbb{N}$ is the coordinate associated with the mth axis. In other words, there exists a unique mapping $n \mapsto (i_1, \ldots, i_M)$. In this way, the graph vertex signal can be viewed as a field in an M-dimensional coordinate system, that is, each vertex can be defined equivalently as $x(n) \equiv x(i_1, \ldots, i_M) \in \mathbb{R}$, that is, it induces the mapping x: $\mathbb{N}^M \mapsto \mathbb{R}$.

When discrete points in the field, x: $\mathbb{N}^M \mapsto \mathbb{R}$, are sampled using a regular lattice of dimensions $I_1 \times \cdots \times I_M$, thereby sampling a total of

$$\prod_{m=1}^{M} I_m \equiv N$$

discrete points, the collection of samples naturally forms the tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_M}$, with its $(i_1, \ldots, i_M)$th entry defined as

$$[\boldsymbol{\mathcal{X}}]_{i_1 \ldots i_M} = x(i_1, \ldots, i_M), \quad i_m \in \mathbb{N}, \ m = 1, 2, \ldots, M. \tag{11.1}$$

Figure 11.1 illustrates a collection of discrete points from a field in a 3-dimensional coordinate system, which together form an order-3 tensor. This procedure is referred to as *tensorization*.

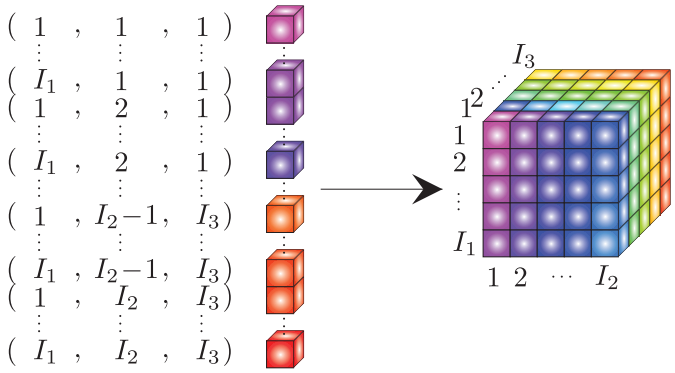


Figure 11.1: Tensorization of discrete samples from a field $x : \mathbb{N}^3 \mapsto \mathbb{R}$.

Remark 24: Real-world examples of a field in M-dimensional coordinates include:

- Netflix ratings in the *user* $\times$ *movie* space ($M = 2$);
- Temperature measurements in the *longitude* $\times$ *latitude* $\times$ *altitude* space ($M = 3$);
- Video pixels in the *time* $\times$ *column* $\times$ *row* $\times$ *RGB* space ($M = 4$);
- EEG signals in the *time* $\times$ *frequency* $\times$ *channel* $\times$ *subject* $\times$ *trial* space ($M = 5$).

11.2 Tensor Decomposition

If the underlying field, x: $\mathbb{N}^M \mapsto \mathbb{R}$, is defined as a *multilinear map* of the form

$$x\colon \underbrace{\mathbb{N} \times \cdots \times \mathbb{N}}_{M \text{ times}} \mapsto \mathbb{R} \tag{11.2}$$

then it is said to be *linearly separable*, and therefore admits the following decomposition

$$x(i_1, \ldots, i_M) = \prod_{m=1}^{M} x_m(i_m). \tag{11.3}$$

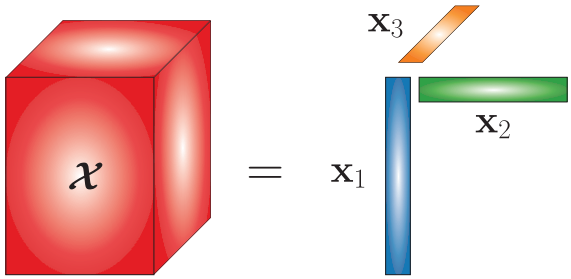


Figure 11.2: Rank-1 CPD of an order-3 tensor.

In other words, the value of $x(i_1, \ldots, i_M)$ is given by the product of M independent single-dimensional functions, x_m: $\mathbb{N} \mapsto \mathbb{R}$, each of which is associated with the mth coordinate axis of the underlying M-dimensional coordinate system. In this way, a tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_M}$, which is sampled from a linearly separable field of the kind in (11.3) admits the following rank-1 canonical polyadic decomposition (CPD)

$$\boldsymbol{\mathcal{X}} = \mathbf{x}_1 \circ \cdots \circ \mathbf{x}_M \tag{11.4}$$

with the symbol $\circ$ denoting the outer product operator, and $\mathbf{x}_m \in \mathbb{R}^{I_m}$ being a parameter vector associated with the mth coordinate axis.

Remark 25: The property in (11.4) is referred to as the *Kronecker separability* condition, which is fundamental to most tensor decompositions and learning algorithms.

With regard to the linear separability property in (11.3), the ith entry of $\mathbf{x}_m$ is given by $[\mathbf{x}_m]_i = x_m(i)$. Figure 11.2 shows the rank-1 CPD of an order-3 tensor.

Kronecker separable tensors admit a vector representation (vectorization), denoted by $\mathbf{x} = \text{vec}(\boldsymbol{\mathcal{X}}) \in \mathbb{R}^N$, which can be expressed as

$$\mathbf{x} = \mathbf{x}_M \otimes \cdots \otimes \mathbf{x}_1 \tag{11.5}$$

and is a direct consequence of (11.4), where the symbol $\otimes$ denotes the Kronecker product operator (see Part I).

Example 36: Consider the data matrix, $\mathbf{X} \in \mathbb{R}^{I \times J}$, which contains the Netflix ratings assigned by I users to J movies, whereby the (i, j)th entry designates the rating assigned by the ith user to the jth movie,

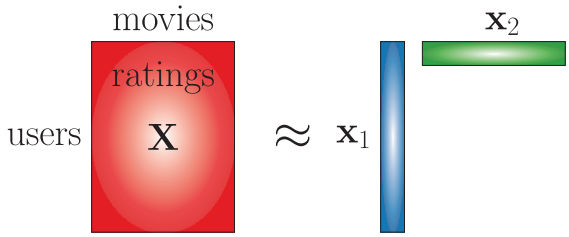


Figure 11.3: Rank-1 CPD of the Netflix rating data matrix.

$x(i,j) \in \mathbb{R}$. The graph representation of this dataset consists of (IJ) vertices residing in a two-dimensional space $(user \times movie)$. Owing to the lattice-like structure of the graph, we can employ its inherent order-2 tensor representation, whereby the data can be approximated using the following rank-1 CPD

$$\mathbf{X} \approx \mathbf{x}_1 \circ \mathbf{x}_2 \equiv \mathbf{x}_1 \mathbf{x}_2^T \tag{11.6}$$

with $\mathbf{x}_1 \in \mathbb{R}^I$ being the factor associated with the *user* axis, and $\mathbf{x}_2 \in \mathbb{R}^J$ the factor associated with the *movie* axis. Note that for order-2 tensors, the CPD is equivalent to the singular value decomposition (SVD). Figure 11.3 illustrates the tensor decomposition of the Netflix rating data matrix.

The factorization of $\mathbf{X}$ assumes that the rating assigned by the ith user to the jth movie can be approximated as

$$x(i,j) \approx x_1(i) x_2(j) \tag{11.7}$$

where $x_1(i) \equiv [\mathbf{x}_1]_i$ and $x_2(j) \equiv [\mathbf{x}_2]_j$. In other words, the rating, $x(i,j)$, can be approximated by a rating assigned by the ith user to all movies, $x_1(i)$, multiplied by a rating assigned to the jth movie by all users, $x_2(j)$.

The so achieved parameter reduction becomes evident, since we have reduced a fully connected (IJ) parameter model to an $(I+J)$ parameter model. This parameter reduction is most pronounced for higher-order tensors, e.g., an order-N tensor model with $\prod_{n=1}^{N} I_n$ parameters (exponential) reduces to a $\sum_{n=1}^{N} I_n$ parameter (linear) model.

11.3 Connectivity of a Tensor

We next show that the tensor structure inherent to $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_M}$ can be modelled naturally as a graph. This is achieved by exploiting the well-known property of lattice-structured graphs which can be decomposed into constituent single-dimensional path graphs.

Building upon the Cartesian product of two disjoint path graphs, as considered in Part I of this monograph, the Cartesian product of M disjoint I_m-vertex path graphs, $\mathcal{G}_m = (\mathcal{V}_m, \mathcal{E}_m)$ for $m = 1, \ldots, M$, yields a graph with an M-dimensional regular lattice structure, denoted by $\mathcal{G} = \mathcal{G}_M \square \cdots \square \mathcal{G}_1 = (\mathcal{V}, \mathcal{B})$, with the symbol $\square$ denoting the graph Cartesian product. In this way, the resulting vertex set takes the form $\mathcal{V} = \mathcal{V}_M \times \cdots \times \mathcal{V}_1$, and the resulting graph contains a total of $\prod_{m=1}^{M} I_m \equiv N$ vertices.

If the adjacency matrix of the mth path graph, $\mathcal{G}_m$, is denoted by $\mathbf{A}_m \in \mathbb{R}^{I_m \times I_m}$, then the adjacency matrix of the resulting M-dimensional regular lattice graph, $\mathcal{G}$, is given by

$$\mathbf{A} = (\mathbf{A}_M \oplus \cdots \oplus \mathbf{A}_1) \in \mathbb{R}^{N \times N} \tag{11.8}$$

where the symbol $\oplus$ denotes the Kronecker sum operator (we refer to Part I). Such an adjacency matrix is said to be *Kronecker summable*.

Remark 26: The adjacency matrix, $\mathbf{A}$, when interpreted through the underlying tensor, describes the connectivity between the entries of vectorization of a tensor, $\mathbf{x} \in \mathbb{R}^N$, while $\mathbf{A}_m \in \mathbb{R}^{I_m \times I_m}$ describes the connectivity between entries along the mth mode. Under this model, the entries of the tensor are only connected to neighboring entries which reside in the same *fiber*.

For illustration purposes, Figure 11.4 shows the Cartesian product of three disjoint path graphs, which results in a graph with a three-dimensional lattice structure. This graph would naturally represent the connectivity between the entries of an order-3 tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{2 \times 3 \times 2}$. Next, consider the order-2 tensor, $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$, with entries sampled from the field, x: $\mathbb{N}^2 \mapsto \mathbb{R}$, using a 2-dimensional regular lattice as illustrated in Figure 11.5.

Example 37: Consider a field on a two-dimensional coordinate system, denoted by x: $\mathbb{N}^2 \mapsto \mathbb{R}$, and illustrated in Figure 11.6. If the scalar field

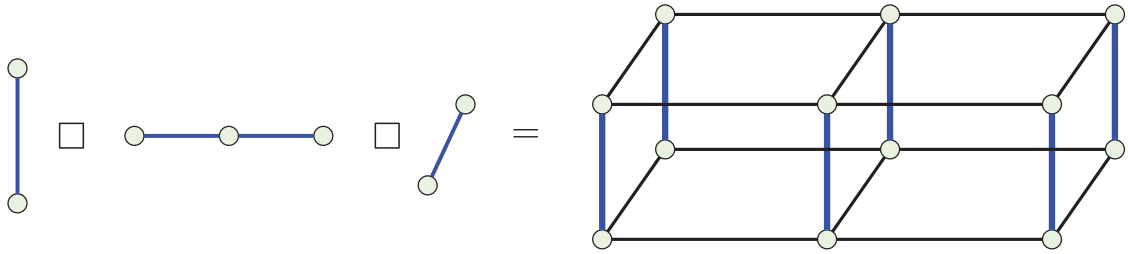


Figure 11.4: Cartesian product of 3 path graphs.

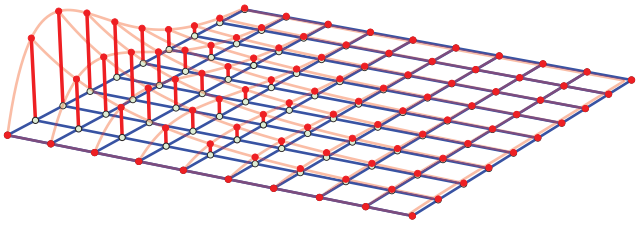


Figure 11.5: Order-2 tensor, $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$, sampled from x: $\mathbb{N}^2 \mapsto \mathbb{R}$.

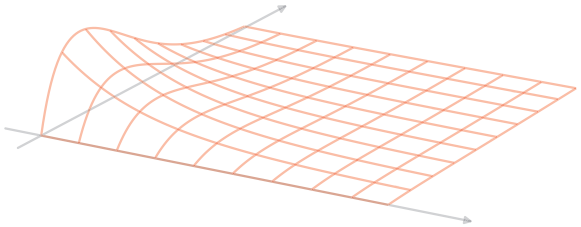


Figure 11.6: An example of a field, x: $\mathbb{N}^2 \mapsto \mathbb{R}$.

is linearly separable, that is, $x(t_1, t_2) = x_1(t_1)x_2(t_2)$, then the sampled tensor, $\mathbf{X}$, is Kronecker separable, and can therefore be expressed as

$$\mathbf{X} = \mathbf{x}_1 \circ \mathbf{x}_2 \iff \text{vec}(\mathbf{X}) = \mathbf{x}_2 \otimes \mathbf{x}_1 \tag{11.9}$$

through $x_1 \in \mathbb{R}^{I_1}$ and $x_2 \in \mathbb{R}^{I_2}$ as data on path graphs sampled respectively from the single-dimensional fields, x_1: $\mathbb{N} \mapsto \mathbb{R}$ and x_2: $\mathbb{N} \mapsto \mathbb{R}$, as illustrated in Figures 11.7–11.8.

11.4 DFT of a Tensor

We have shown in expression (11.8) above that tensors can be considered as a special class of graphs which exhibit a Kronecker summable adjacency matrix. In that case, the DFT of a tensor can be naturally

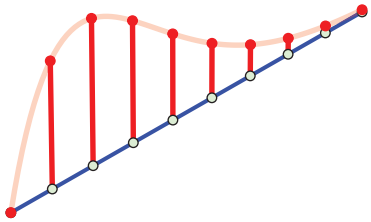


Figure 11.7: Path graph signal, $\mathbf{x}_1 \in \mathbb{R}^{I_1}$, sampled from x_1: $\mathbb{N} \mapsto \mathbb{R}$.

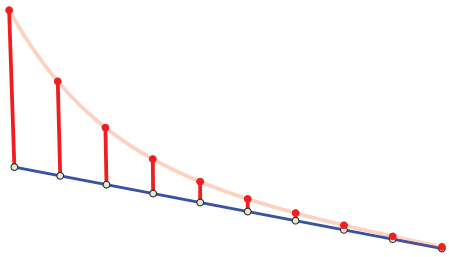


Figure 11.8: Path graph signal, $\mathbf{x}_2 \in \mathbb{R}^{I_2}$, sampled from x_2: $\mathbb{N} \mapsto \mathbb{R}$.

obtained from the graph Fourier transform (GFT), which was introduced in Part II of this monograph. In this way, the GFT of a graph with a lattice structure can be performed by evaluating the eigenvalue decomposition of the adjacency matrix $\mathbf{A}$, given by

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1} \tag{11.10}$$

where $\mathbf{U} \in \mathbb{R}^{N \times N}$ and $\mathbf{\Lambda} \in \mathbb{R}^{N \times N}$ denote respectively the matrices of eigenvectors and eigenvalues of $\mathbf{A}$.

Owing to the Kronecker sum structure of $\mathbf{A}$ in (11.8), the eigenvector and eigenvalue matrices of GFT exhibit the following structure

$$\mathbf{U} = (\mathbf{U}_M \otimes \cdots \otimes \mathbf{U}_1) \tag{11.11}$$

$$\mathbf{\Lambda} = (\mathbf{\Lambda}_M \oplus \cdots \oplus \mathbf{\Lambda}_1) \tag{11.12}$$

where $\mathbf{U}_m \in \mathbb{R}^{I_m \times I_m}$ and $\mathbf{\Lambda} \in \mathbb{R}^{I_m \times I_m}$ respectively denote the matrices of eigenvectors and eigenvalues of the mth path graph adjacency matrix, $\mathbf{A}_m$, obtained through

$$\mathbf{A}_m = \mathbf{U}_m \mathbf{\Lambda}_m \mathbf{U}_m^{-1}. \tag{11.13}$$

Therefore, the eigenvectors of $\mathbf{A}$ are said to be *Kronecker separable*, while the eigenvalues are *Kronecker summable*.

11.5 Unstructured Graphs

Consider an N-vertex graph, $\mathcal{G}$, with vertex signals sampled from the field, x: $\mathbb{R}^M \mapsto \mathbb{R}$, using a regular lattice, which together form the order-M tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_M}$, with $\prod_{m=1}^{M} I_m \equiv N$.

Similarly, consider a K-vertex graph, $\tilde{\mathcal{G}}$, with vertex signals also sampled from the same field, x: $\mathbb{R}^M \mapsto \mathbb{R}$, but using instead an unstructured sampling scheme. In this way, the unstructured graph can be defined as a subset of a lattice-structured graph, i.e., $\tilde{\mathcal{G}} \subset \mathcal{G}$.

The vertex signals of $\tilde{\mathcal{G}}$, denoted by the vector $\tilde{\mathbf{x}} \in \mathbb{R}^K$, can therefore be defined as

$$\tilde{\mathbf{x}} = \boldsymbol{\Pi}\text{vec}(\boldsymbol{\mathcal{X}}) \tag{11.14}$$

where $\boldsymbol{\Pi} \in \mathbb{R}^{K \times N}$ is a *sampling matrix*, with entries defined as

$$[\boldsymbol{\Pi}]_{kn} = \begin{cases} 1, & \text{if } \tilde{x}(k) \equiv x(n), \\ 0, & \text{otherwise} \end{cases} \tag{11.15}$$

with $\tilde{x}(k) \in \mathbb{R}$ and $x(n) \in \mathbb{R}$ denoting respectively the kth vertex of $\tilde{\mathcal{G}}$ and the nth vertex of $\mathcal{G}$.

Although the lattice-structured graph, $\mathcal{G}$, exhibits a Kronecker separable signal vector and a Kronecker summable adjacency matrix, the associated unstructured graph, $\tilde{\mathcal{G}}$, does not have such properties because, in general, $\boldsymbol{\Pi}$ is not separable. This can be seen from the relationship between the adjacency matrices of $\tilde{\mathcal{G}}$ and $\mathcal{G}$, which is given by

$$\tilde{\mathbf{A}} = \boldsymbol{\Pi}\mathbf{A}\boldsymbol{\Pi}^T = \boldsymbol{\Pi}(\mathbf{A}_M \oplus \cdots \oplus \mathbf{A}_1)\boldsymbol{\Pi}^T. \tag{11.16}$$

Notice that the last term above cannot be decomposed further if $\boldsymbol{\Pi}$ is not separable. A direct consequence of the result in (11.16) is that the GFT bases of $\tilde{\mathcal{G}}$ (eigenvalue decomposition of $\tilde{\mathbf{A}}$) do not exhibit the Kronecker summability either.

Example 38: Referring back to Example 37, the graph signal resulting from an irregular sampling of the field x: $\mathbb{R}^2 \mapsto \mathbb{R}$ is not Kronecker separable as it cannot be represented as a Cartesian product of two path graphs (as in Figures 11.5–11.8), as illustrated in Figure 11.9.

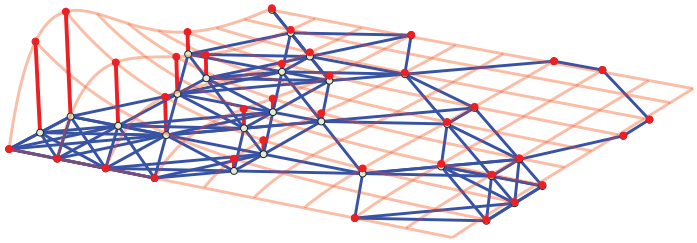


Figure 11.9: Unstructured graph, $\tilde{\mathbf{x}} \in \mathbb{R}^K$, sampled from x: $\mathbb{N}^2 \mapsto \mathbb{R}$.

11.6 Tensor Representation of Multi-Relational Graphs

The rapidly growing prominence of multi-relational network data in areas as diverse as social network modeling, the semantic web, bioinformatics and artificial intelligence, has brought to light the increasing importance of Data Analytics on domains where the entities are interconnected by multiple relations. To put this into context of graphs, while traditional graph models only account for a single relation type, designated by the adjacency matrix, $\mathbf{A} \in \mathbb{R}^{N \times N}$, a multi-relational N-vertex graph may exhibit a large number, say M, of distinct relation types between vertices. In this case, a multi-relational graph would be defined by M adjacency matrices, $\mathbf{A}_m \in \mathbb{R}^{N \times N}$ for $m = 1, \ldots, M$; one for each relation type.

While it is possible to model this situation through a short and wide $N \times MN$ dimensional matrix, this would both involve numerical difficulties and obscure physical relevance. To this end, to model such a multi-relational graph in a parsimonious and compact manner, we may construct a three-way tensor, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{N \times N \times M}$, whereby its mth frontal slice is given by $\mathbf{A}_m$. In this way, the first two modes define the entity domain, while the third mode represents the relation domain, as illustrated in Figure 11.10. The tensor entry $[\boldsymbol{\mathcal{A}}]_{ijk} = 1$ therefore designates the existence of a relation between the ith and jth entities within the kth relation type; otherwise, for non-existing and unknown relations, the entry is set to zero.

The work in Lin *et al.* (2008, 2009), Tang *et al.* (2009), Nickel *et al.* (2011), Papalexakis *et al.* (2013), Gauvin *et al.* (2014), Verma and Bharadwaj (2017a), Verma and Bharadwaj (2017b), and Katsimpras and Paliouras (2020) employs such tensor model to learn an inherent

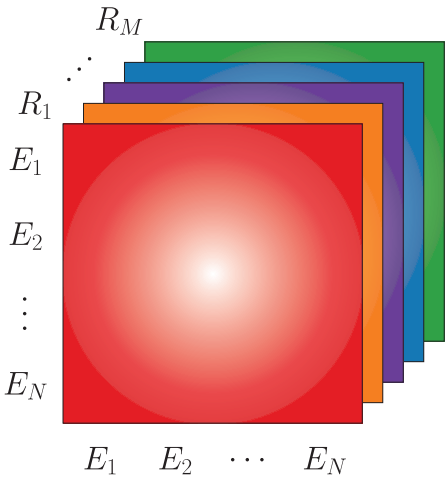


Figure 11.10: Construction of a multi-relational adjacency tensor, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{N \times N \times M}$, where E_n denotes the nth entity and R_m the mth relation type.

structure from multi-relational data. The following rank-L factorization was employed, known as the RESCAL decomposition (Nickel *et al.*, 2011), whereby each frontal slice of $\boldsymbol{\mathcal{A}}$ is factorized as

$$\mathbf{A}_m = \mathbf{U}\mathbf{R}_m\mathbf{U}^T, \quad m = 1, \ldots, M \tag{11.17}$$

where $\mathbf{U} \in \mathbb{R}^{N \times L}$ is a factor matrix which maps the N-dimensional entity space to an L-dimensional latent component space, and $\mathbf{R}_m \in \mathbb{R}^{L \times L}$ models the interactions of latent components within the mth relation type. Alternatively, this can be expressed in terms of the factorization of the tensor $\boldsymbol{\mathcal{A}}$, in the form

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{R}} \times_1 \mathbf{U} \times_2 \mathbf{U} \tag{11.18}$$

where the symbol $\times_n$ denotes the mode-n product, and $\boldsymbol{\mathcal{R}} \in \mathbb{R}^{L \times L \times M}$ is the latent core tensor with $\mathbf{R}_m$ being its mth frontal slice, as illustrated in Figure 11.11. Such a factorization allows for link-based clustering, whereby the entities $E_1, \ldots, E_N$ are clustered according to the information in $\mathbf{U}$ only. In doing so, the similarity between entities is computed based on their similarity across multiple relations.

Example 39: Social networks play an important role in the functionality of an organization and it is therefore of considerable interest to analyze the properties of such networks. The adoption of social networking services within organizations can largely facilitate the interaction and

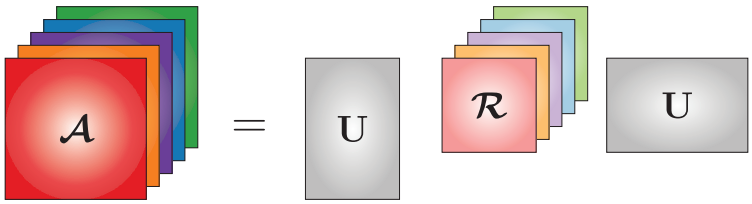


Figure 11.11: Factorization of a multi-relational adjacency tensor, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{N \times N \times M}$ as in (11.17).

collaboration between employees. For example, a social network could reveal information about the characteristics of an employee which could then be used to improve efficiency and influence team structuring.

As shown in Figure 6.8 and Example 26, a social network can be modelled as a graph, whereby each vertex represents an individual (employee) and each edge designates the existence of a social relationship between two individuals. While a conventional graph can model social networks involving one type of relationship, multi-relational graphs allow for the modelling of multiple (and different) types of relationships. Figure 11.12 illustrates a multi-relational social network involving three employees (vertices) who communicate via email (blue edge), LinkedIn (green edge) and Skype (orange edge). Observe that social relationships may be directed, e.g., employee A sends emails (blue edge) to employee B but not vice versa. If the adjacency matrix associated with the mth relationship type is given by $\mathbf{A}_m \in \mathbb{R}^{3\times 3}$ for $m = 1, 2, 3, 4$, then the adjacency tensor, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{3\times 3\times 4}$, can be constructed to model the entire social network. Once the latent components (factor) matrix, $\mathbf{U} \in \mathbb{R}^{3\times L}$, is inferred from $\boldsymbol{\mathcal{A}}$ using the factorization in (11.17), it is possible to apply feature-based clustering to obtain the inherent community structure in such multi-relational network. The output of this step would be a set of K disjoint communities (sub-graphs), $\{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$.

11.7 Multi-Graph Tensor Networks

After exploring the inherent links between graphs and tensors, in Section 11.6, and between graphs and neural networks, in Section 10, it is natural to further explore the possibilities enabled by a joint consideration of these three domains. One such general approach is referred to

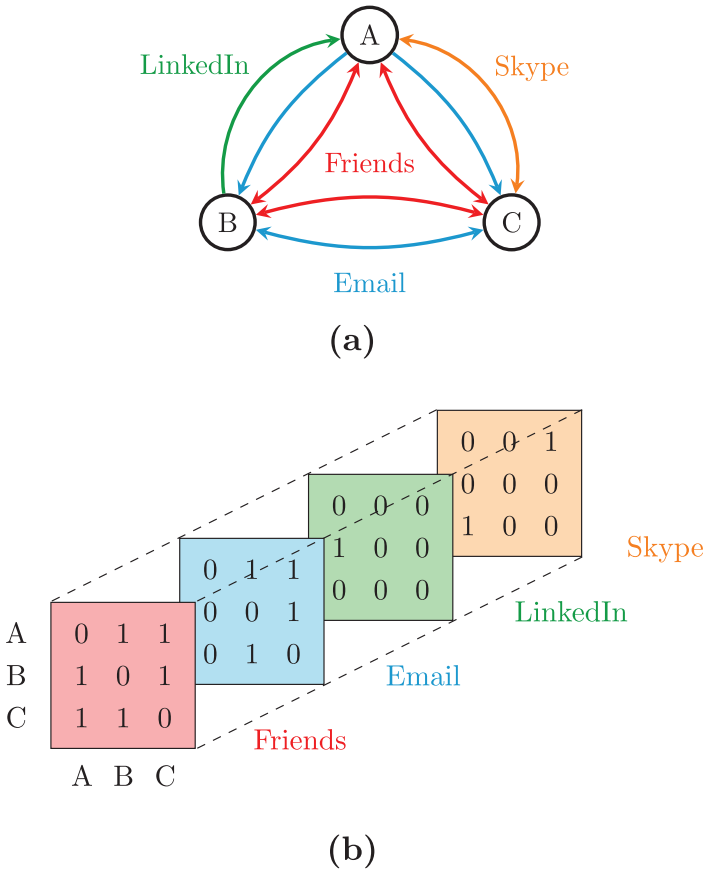


Figure 11.12: Social network modelled as a multi-relational graph. (a) Graph representation of the social network. (b) Adjacency tensor, $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{3\times3\times4}$, associated with the social network in (a).

as the Multi-Graph Tensor Network (MGTN) model (Xu *et al.*, 2020), which aims to fully exploit the virtues of both graphs and tensors in a deep learning setting. For a joint account between tensors and neural networks, we refer to Calvi *et al.* (2019) and Bacciu and Mandic (2020). In this way, the MGTN framework is capable of:

- Handling irregular data that reside on multiple graph domains;
- Leveraging on the compression and structure-preserving properties of tensor networks, to enhance the expressive power of NNs, at a reduced parameter complexity.

The MGTN generalises the Recurrent Graph Tensor Network (RGTN) model, introduced in Xu and Mandic (2020), which enables deep modelling on irregular domains and was developed with the aim

to model time-series problems related to sequential data, and was only defined for a single graph domain. To make this concept suitable for applications beyond time-series and in a Big Data setting, the MGTN operates in a multi-modal data setting defined on multiple graph domains and thus not limited to time-series.

More precisely, the time-based multi-linear graph filter, $\boldsymbol{\mathcal{R}}$, which underpins the RGTN in Xu and Mandic (2020), employs a time-graph adjacency matrix that reflects the temporal flow of information. On the other hand, for a given weighted graph adjacency matrix, $\mathbf{A} \in \mathbb{R}^{I_1 \times I_1}$, the MGTN approach constructs a multi-linear graph filter in the tensor domain, $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{J_1 \times I_1 \times J_1 \times I_1}$, given by

$$\boldsymbol{\mathcal{F}} = \text{ten}(\mathbf{I} + (\mathbf{A} \otimes \mathbf{P})) \tag{11.19}$$

where the propagation matrix, $\mathbf{P} \in \mathbb{R}^{J_1 \times J_1}$, models the flow of information between neighboring vertices (as opposed to successive time-steps in the RGTN case) and the operator $\text{ten}(\cdot)$ represents a suitable tensorization, as, for example, that in Figure 11.1. This allows us to adapt the multi-linear graph filter, $\boldsymbol{\mathcal{F}}$, to any given graph domain of any data modality.

Consider a general multi-graph learning problem where the input is an order-$(M+1)$ tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{J_0 \times I_1 \times I_2 \times \cdots \times I_M}$, with J_0 features indexed along M physical modes $\{I_1, I_2, \ldots, I_M\}$, such that a graph, $\mathcal{G}^{(m)}$, is associated with each of the I_m modes, $m = 1, \ldots, M$. For this problem, we can define:

1. $\mathcal{A} = \{\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(M)}\}$, a set of adjacency matrices, $\mathbf{A}^{(m)} \in \mathbb{R}^{I_m \times I_m}$, constructed from the corresponding graphs $\mathcal{G}^{(m)}$.

2. $\mathcal{W} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \ldots, \mathbf{W}^{(M)}\}$, a set of weight matrices, $\mathbf{W}^{(m)} \in \mathbb{R}^{J_m \times J_{m-1}}$, used for feature transforms, where J_m, for $m = 1, \ldots, M$, controls the number of feature maps at every mode m.

3. $\mathcal{P} = \{\mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \ldots, \mathbf{P}^{(M)}\}$, a set of propagation matrices, $\mathbf{P}^{(m)} \in \mathbb{R}^{J_m \times J_m}$, modelling the propagation of information over the neighbors of the graph $\mathcal{G}^{(m)}$.

The general *Multi-Graph Tensor Network* (gMGTN) layer is characterized by the forward pass

$$\begin{aligned}\boldsymbol{\mathcal{Y}} = \phi(\boldsymbol{\mathcal{F}}^{(M)} \times_{3,4}^{1,M+1} \mathbf{W}^{(M)} \times_2^1 \cdots \times_2^1 \boldsymbol{\mathcal{F}}^{(2)} \times_{3,4}^{1,3} \mathbf{W}^{(2)} \\ \times_2^1 \boldsymbol{\mathcal{F}}^{(1)} \times_{3,4}^{1,2} \mathbf{W}^{(1)} \times_2^1 \boldsymbol{\mathcal{X}})\end{aligned} \tag{11.20}$$

where $\phi(\cdot)$ is an optional non-linear activation function and $\boldsymbol{\mathcal{F}}^{(m)} = \text{ten}(\mathbf{I} + (\mathbf{A}^{(m)} \otimes \mathbf{P}^{(m)}))$. The so defined forward pass generates a feature map, $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_M \times I_1 \times \cdots \times I_M}$, from the input tensor, $\boldsymbol{\mathcal{X}}$, through a series of multi-linear graph filter and weight matrix contractions, which essentially iterates the graph filtering operation across all M graph domains.

Since the gMGTN learns a propagation matrix, $\mathbf{P}^{(m)}$, and a weight matrix, $\mathbf{W}^{(m)}$, for each of the M graphs, when $J_1 = J_2 = \cdots = J_M = J$, this results in a parameter complexity of $\mathcal{O}(MJ^2)$, which is linear in the number of graphs, M, but quadratic in the size of feature maps, J, so that the computation quickly becomes intractable for high dimensional multi-graph problems.

The computational bottleneck can be resolved by approximating $\mathbf{P}^{(m)} \approx \mathbf{I}$ for $m = 1, \ldots, M$, and by using a single weight matrix, $\mathbf{W}^{(x)} \in \mathbb{R}^{J_1 \times J_0}$, for all of the graph domains, where J_1 controls the number of hidden units (feature maps).

The resulting fast MGTN (fMGTN) is shown in Figure 11.13, and exhibits the following reduced forward pass

$$\boldsymbol{\mathcal{Y}} = \phi(\mathbf{F}^{(M)} \times_2^{M+1} \cdots \times_2^4 \mathbf{F}^{(2)} \times_2^3 \mathbf{F}^{(1)} \times_2^2 \mathbf{W}^{(1)} \times_2^1 \boldsymbol{\mathcal{X}}) \tag{11.21}$$

where $\mathbf{F}^{(m)} = (\mathbf{I} + \mathbf{A}^{(m)})$ is a standard graph shift filter. As the fMGTN does not have to learn $\mathbf{P}^{(m)}$ or $\mathbf{W}^{(m)}$, the parameter complexity of the forward pass is reduced from $\mathcal{O}(MJ^2)$ to $\mathcal{O}(J^2)$ but at the cost of lower expressive power. After extracting the feature map, $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_1 \times I_1 \times \cdots \times I_M}$, it is customary to flatten the extracted features into a vector, in order to pass them through dense neural network layers to generate the fMGTN output. To further reduce parameter complexity, the weight matrices of the dense layers can be tensorized and represented in some compressed tensor format, as discussed in Novikov *et al.* (2015), Cichocki *et al.* (2016, 2017), Calvi *et al.* (2019). This not only further reduces

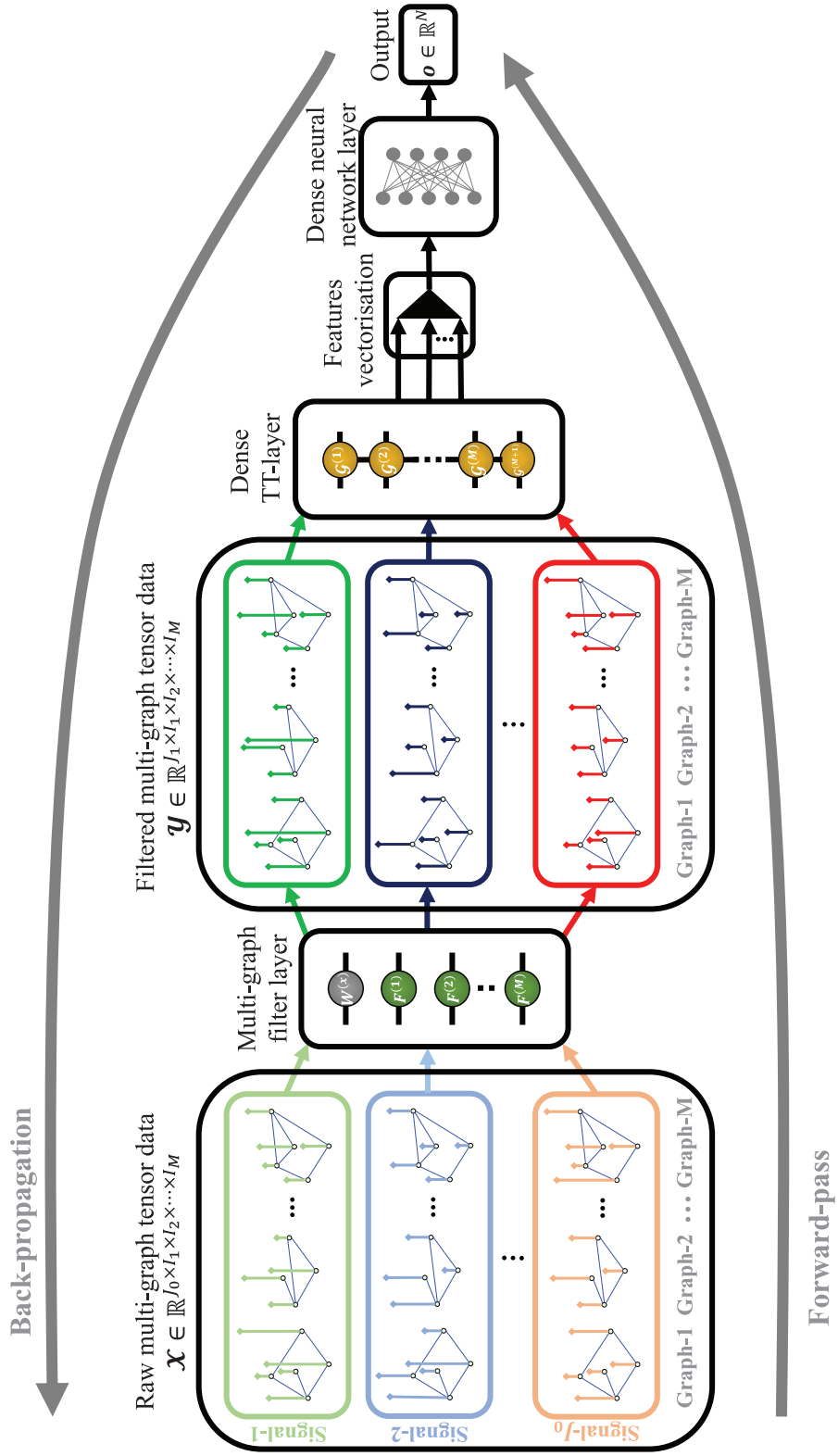


Figure 11.13: Illustration on the structure of a fast multi-graph tensor network.

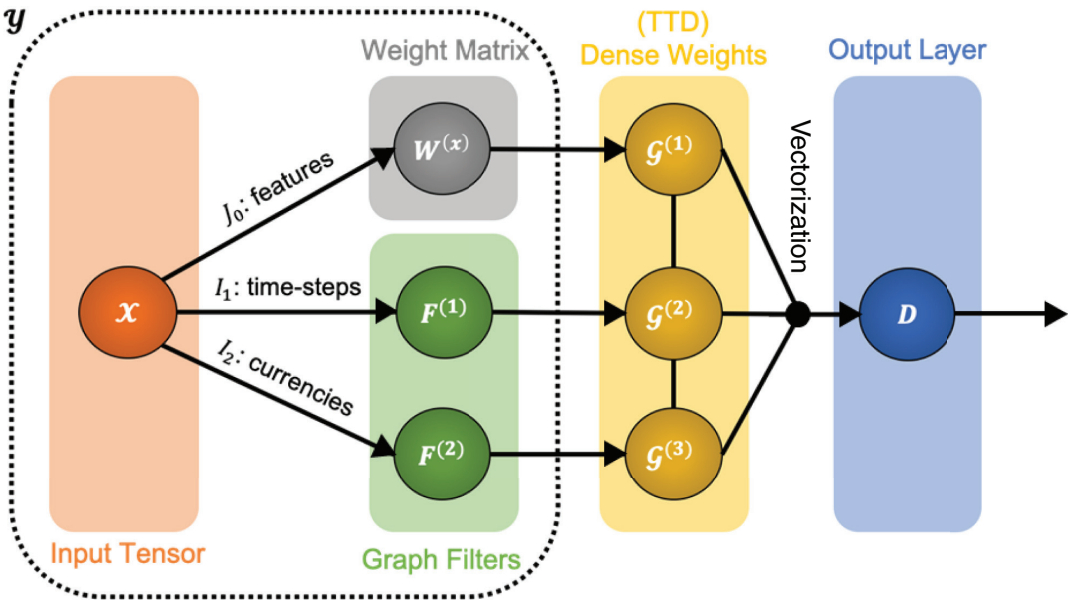


Figure 11.14: Tensor network representation of the fast Multi-Graph Tensor Network (fMGTN) used in Example 40 and shown in Figure 11.13. The section encircled in dotted line denotes the multi-graph filtering operation for $M = 2$ as in (11.21). The yellow region designates a tensorized dense layer weight matrix, represented in the Tensor-Train format (TTD). The input data used for the experiment is an order-3 tensor of FOREX data, with J_0 pricing features, I_1 past time-steps, and I_2 currencies. Note that as input data modes, this MGTN employs a time-domain graph filter of dimension I_1 and a currency-domain graph filter of dimension I_2.

the number of parameters, but also maintains compatibility with the inherent multi-modal nature of the problem.

Example 40: We considered the task of Foreign Exchange (FOREX) algorithmic trading in order to illustrate the possibilities enabled by the MGTN framework. A conceptual application of the combination of graphs, tensors and neural networks in this scenario is shown in Figure 11.14.

The MGTN setting is general enough to be applicable in a range of other domains, including social networks, communication networks, and cognitive neuroscience.

12

Metro Traffic Modeling Through Graphs

With the rapid development of many economies, an increasing proportion of the world's population moving to cities, urban traffic congestion is becoming a serious issue. For example, underground traffic networks routinely undergo general maintenance, frequently exhibit signal failures and train derailments, and may even occasionally experience emergency measures because of various accidents. Such events ultimately require the closure of at least one station which may severely impact the service across the entire network. The economic costs of these transport delays to central London business are estimated to be £1.2 billion per year. Hence, appropriate and physically meaningful tools to understand, quantify, and plan for the resilience of these traffic networks to disruptions are much needed.

In this section, we demonstrate how the concept of vertex centrality of an adjacency matrix (for more detail, see Part I of this monograph) may be employed to identify those stations in the London underground network which have the greatest influence on the functionality of the traffic, and proceed, in an innovative way, to assess the impact of a station closure on service levels across the city. Such underground network vulnerability analysis offers the opportunity to analyze, optimize

and enhance the connectivity of the London underground network in a mathematically tractable and physically meaningful manner.

12.1 Traffic Centrality as a Graph-Theoretic Measure

The underground network can be modelled as an undirected N-vertex graph, denoted by $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, with $\mathcal{V}$ as the set of N vertices (stations) and $\mathcal{E}$ the set of edges (underground lines) connecting the vertices (stations) (Dees *et al.*, 2019). The connectivity of the network is encoded within the (undirected) adjacency matrix, $\mathbf{A} \in \mathbb{R}^{N \times N}$. Figure 12.1 illustrates the proposed graph model of the London underground network, with each vertex representing a station, and each edge designating the underground line connecting two adjacent stations. Notice that standard data analytics domains are ill-equipped to deal with this class of problems.

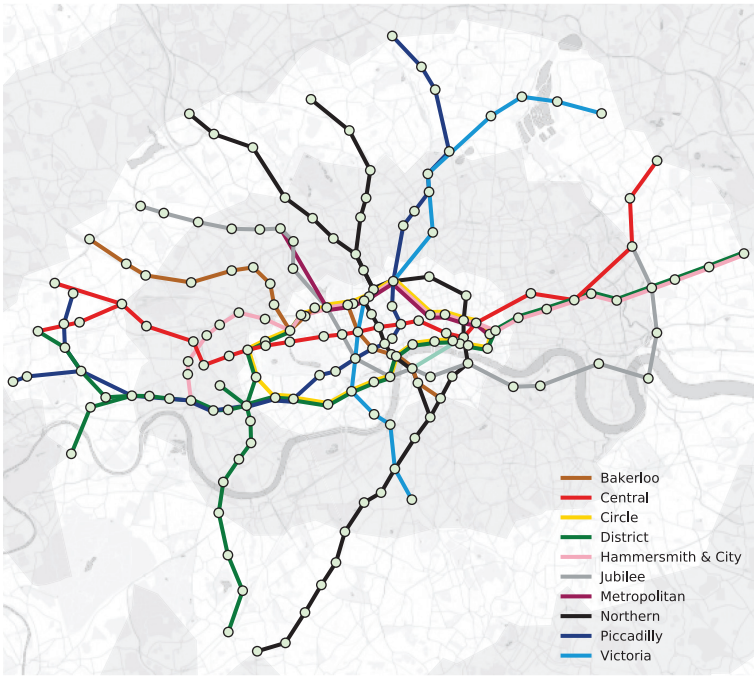


Figure 12.1: Graph model of the London underground network in Zones 1–3.

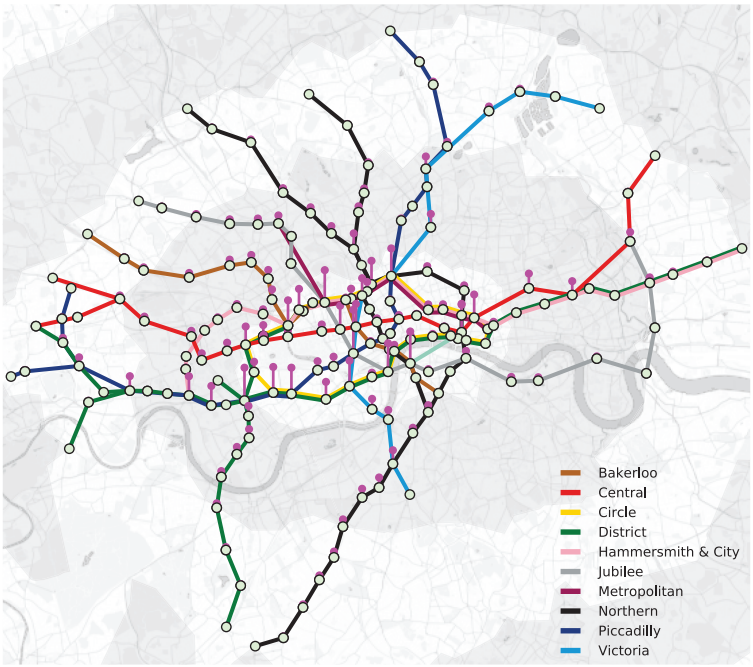


Figure 12.2: Betweenness centrality, designated by magenta-colored bars, of the London underground network in Zones 1–3. The largest betweenness centrality is observed for the following stations: Green Park, Earl's Court, Baker Street, Waterloo and Westminster.

We employ the following metrics to characterize the topology of the network and model its vulnerability.

- *Betweenness centrality*, which reflects the extent to which a given vertex lies in between pairs or groups of other vertices of the graph, and is given by

$$B_n = \sum_{k,m \in \mathcal{V}} \frac{\sigma(k, m \mid n)}{\sigma(k, m)} \tag{12.1}$$

  where $\sigma(k, m)$ denotes the number of shortest paths between vertices k and m, and $\sigma(k, m \mid n)$ the number of those paths passing through vertex n (Freeman, 1977). In terms of the actual metro traffic, this can also be interpreted as the extent to which a vertex is an *intermediate in the communication over the network*. Figure 12.2 shows that, as expected, the stations at the center of the city exhibit the largest betweenness centrality, and their disconnection would therefore severely impact the communication over the underground network.

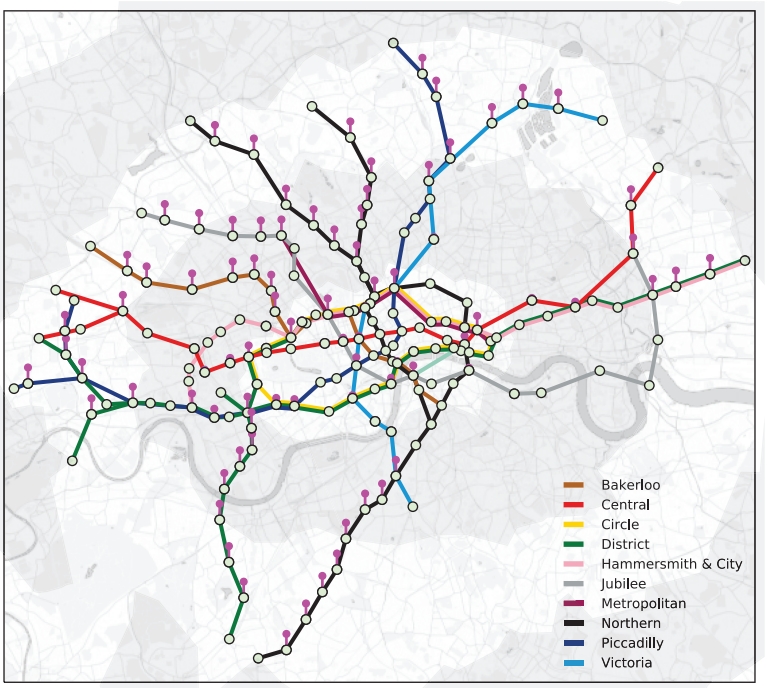


Figure 12.3: Closeness vitality, designated in magenta bars, of the London underground network in Zones 1–3.

- *Closeness vitality*, which represents the change in the sum of distances between all vertex pairs after excluding the nth vertex (Brandes, 2005). Figure 12.3 shows that the stations located in the more remote areas of Zones 2–3 exhibit the largest closeness vitality measure. This is because their removal from the network would disconnect the stations located at the boundaries from the rest of the network.

12.2 Modeling Commuter Population from Net Passenger Flow

In this section, we employ graph theory to analyze the net passenger flow at all stations of the London underground network. In particular, we demonstrate that it is possible to infer the resident population surrounding each station based on the net passenger flow during the morning rush hour alone (Dees *et al.*, 2019).

To derive the corresponding graph model, we employed the *Fick law of diffusion* (closely related to Newton's law of cooling discussed in Section 10.2 and Laplacian diffusion maps described in Part I) which

relates the diffusive flux to the concentration of a given vector field, under a steady state assumption. This model asserts that the flux flows from regions of high concentration (population) to regions of low concentration (population), with a magnitude that is proportional to the concentration gradient. Mathematically, the Fick law is given by

$$\mathbf{q} = -k\nabla\boldsymbol{\phi} \tag{12.2}$$

where

- $\mathbf{q}$ is the flux which measures the amount of substance per unit area per unit time (mol m^{-2} s^{-1});
- k is the coefficient of diffusivity, with its value equal to area per unit time (m^2 s^{-1});
- $\boldsymbol{\phi}$ represents the concentration (mol m^{-3}).

In this way, we can model the passenger flows in the London underground network as a diffusion process, whereby during the morning rush hour the population mainly flows from concentrated residential areas to sparsely populated business districts. Therefore, the variables in our model are:

- $\mathbf{q} \in \mathbb{R}^N$, the net passenger flow vector, where the ith entry represents the net passenger flow at the ith station during the morning rush hour, that is

$$\begin{aligned} q(i) = &\,(\text{passengers exiting station } i) \\ &- (\text{passengers entering station } i) \end{aligned} \tag{12.3}$$

  with its value equal to "passengers per station per unit time";
- $k = 1$, the coefficient of diffusivity, with its dimension equal to "stations per unit time";
- $\boldsymbol{\phi} \in \mathbb{R}^N$, the resident population in the area surrounding the station.

This model therefore suggests that, in the morning, the net passenger flow at the ith station, $q(i)$, is proportional to the population difference between the areas surrounding a station i and the adjacent stations j, that is

$$\begin{aligned} q(i) &= -k \sum_j A_{ij}(\phi(i) - \phi(j)) \\ &= -k\left(\phi(i)\sum_j A_{ij} - \sum_j A_{ij}\phi(j)\right) \\ &= -k\left(\phi(i)D_{ii} - \sum_j A_{ij}\phi(j)\right) \\ &= -k\sum_j (\delta_{ij}D_{ii} - A_{ij})\phi(j) = -k\sum_j L_{ij}\phi(j). \end{aligned} \tag{12.4}$$

When considering N stations together, the above model assumes the matrix form

$$\mathbf{q} = -k\mathbf{L}\boldsymbol{\phi} \tag{12.5}$$

where $\mathbf{L} = (\mathbf{D} - \mathbf{A}) \in \mathbb{R}^{N\times N}$ is the Laplacian matrix of the graph model (see Part I of this monograph). For clarity, Figure 12.4 illustrates a signal within this diffusion model on a 2-vertex path graph obeying the Fick law.

The data for the average daily net flow of passengers during the morning rush hour at each station in 2016 was obtained from Transport

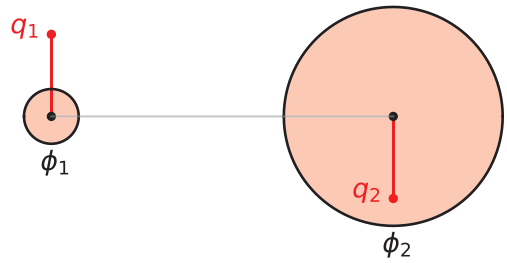


Figure 12.4: Towards a graph representation of the London underground network. A simplified path graph with two stations surrounded by the respective populations, $\phi(1)$ and $\phi(2)$, exhibits the corresponding net fluxes, $q(1)$ and $q(2)$. Intuitively, stations surrounded by large populations experience net in-flows of passengers, whereas stations surrounded by low populations experience net out-flows of passengers.

Table 12.1: Daily average passenger flows during the morning rush hour per transportation Zone in London

Zone	Entries	Exits	Net Outflow
1	455,704	844,123	388,419
2	343,145	264,732	−78,413
3	275,965	104,414	−171,551
4–10	206,408	72,152	−134,256
Total	1,281,222	1,285,421	4,199

for London (TFL) (Transport for London, n.d.), and is plotted as a signal on the graph model of the London underground in Figure 12.5. For illustration purposes, Table 12.1 shows the daily average net flow of passengers per transportation zone. As expected, Zone 1 is the only zone to exhibit a net outflow of passengers, while Zones 2–10 show a net inflow of passengers. In particular, Zone 3 exhibits the largest inflow. In an ideal scenario, the total net outflow across Zones 1–10 should sum up to 0, however, the residual net outflow is attributed to passengers entering the underground network through other transport services not considered in our model, for example, rail services.

Moreover, Table 12.2 shows the average net flow of passengers for the top 5 stations with the greater net inflow and outflow. The stations with the greatest net outflow of passengers are located within the financial (Bank, Canary Wharf, Green Park) and commercial (Oxford Circus, Holborn) districts. In contrast, the greatest net inflow of passengers is attributed to the contribution from the railway stations located in residential areas.

To obtain an estimate of the resident population surrounding each station, we can simply rearrange the passenger flow in (12.5) to obtain

$$\hat{\boldsymbol{\phi}} = -\frac{1}{k}\mathbf{L}^{+}\mathbf{q} \tag{12.6}$$

where the symbol $(\cdot)^{+}$ denotes the matrix pseudo-inverse operator. However, notice that the population vector can only be estimated up to a constant, hence the vector $\hat{\boldsymbol{\phi}}$ actually quantifies the *relative* population

Table 12.2: Stations in the London underground system with greatest net passenger outflow and inflow during the morning rush hour

Zone	Entries	Exits	Net Outflow
Bank	17,577	69,972	52,395
Canary Wharf	8,850	56,256	47,406
Oxford Circus	3,005	44,891	41,886
Green Park	2,370	30,620	28,250
Holborn	1,599	25,294	23,695
Finsbury Park	20,773	8,070	−12,703
Canada Water	31,815	14,862	−16,953
Brixton	24,750	4,369	−20,381
Stratford	43,473	22,360	−21,113
Waterloo	61,129	22,861	−38,268

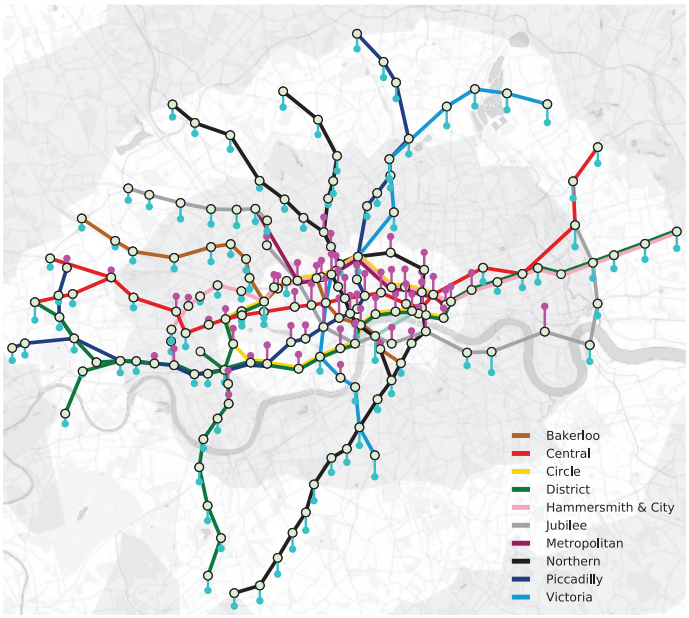


Figure 12.5: Net passenger outflow during the morning rush hour within Zones 1–3 of the London underground network. The magenta bars designate a net outflow of passengers while the cyan bars designate a net inflow of passengers. Stations located within business districts exhibit the greatest net outflow of passengers, while stations located in residential areas, toward the boundaries of Zones 2–3, exhibit the largest net inflow of passengers.

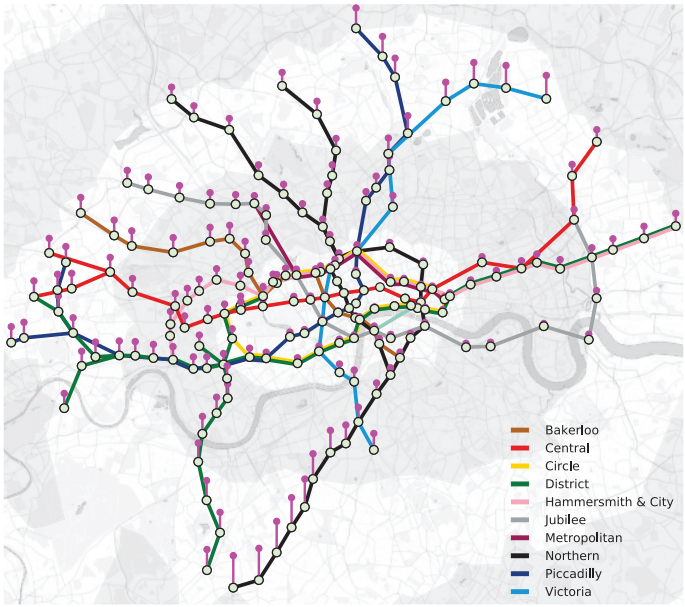


Figure 12.6: Population distribution implied by our graph model in (12.6), calculated from the net passenger outflow during the morning rush hour within Zones 1–3 of London underground system. As expected, business districts exhibit the lowest population density, while residential areas (Zones 2–3) exhibit the highest commuter population density.

between stations, whereby the station with the lowest estimated surrounding population takes the value of 0. The so estimated resident population surrounding each station, based on the morning net passenger flow, is displayed in Figure 12.6 as a signal on a graph. Observe that these estimates are reasonable and physically meaningful since most of the resident population in London is concentrated toward the more remote areas of Zones 2–3, while business districts at the center of Zone 1 are sparsely populated in the evening.

13

Portfolio Cuts

Investment returns naturally reside on irregular domains, however, standard multivariate portfolio optimization methods are agnostic to data structure. To this end, we investigate ways for the domain knowledge to be meaningfully incorporated into the analysis, by means of *portfolio cuts*. Such a graph-theoretic portfolio partitioning technique would allow the investor to devise robust and tractable asset allocation schemes, by virtue of a rigorous graph framework for considering smaller, computationally feasible, and economically meaningful clusters of assets, based on graph cuts. In turn, this makes it possible to fully utilize the covariance matrix of asset returns for constructing the portfolio, even without the requirement for its inversion.

Modern portfolio theory suggests an optimal strategy for minimizing the investment risk, which is based on the second-order moments of asset returns (Markowitz, 1952). The solution to this optimization task is referred to as the *minimum-variance* (MV) portfolio. Consider the vector, $\mathbf{r}(t) \in \mathbb{R}^N$, which contains the returns of N assets at a time t, the ith entry of which is given by

$$r_t(i) = \frac{p_t(i) - p_{t-1}(i)}{p_{t-1}(i)} \tag{13.1}$$

where $p_t(i)$ denotes the value of the ith asset at a time t. The MV portfolio asserts that the optimal vector of asset holdings, $\mathbf{w} \in \mathbb{R}^N$, is obtained through the following optimization problem

$$\min_{\mathbf{w}} \mathbf{w}^T \mathbf{\Sigma} \mathbf{w}, \quad \text{subject to } \mathbf{w}^T \mathbf{1} = 1 \tag{13.2}$$

where $\mathbf{\Sigma} = \text{cov}\{\mathbf{r}\} \in \mathbb{R}^{N \times N}$ is the covariance matrix of returns, $\mathbf{1} = [1, \ldots, 1]^T$, and the constraint, $\mathbf{w}^T \mathbf{1} = 1$, enforces full investment of the capital. The optimal portfolio holdings (using the method of Lagrange multipliers) then become

$$\mathbf{w} = \frac{\mathbf{\Sigma}^{-1} \mathbf{1}}{\mathbf{1}^T \mathbf{\Sigma}^{-1} \mathbf{1}}. \tag{13.3}$$

It is important to highlight that the matrix inversion of $\mathbf{\Sigma}$ required in (13.3) may lead to significant errors for ill-conditioned matrices. These instability concerns have received substantial attention in recent years (Kolm *et al.*, 2014), and alternative procedures have been proposed to promote robustness by either incorporating additional portfolio constraints (Clarke *et al.*, 2002), introducing Bayesian priors (Black and Litterman, 1992) or improving the numerical stability of covariance matrix inversion (Ledoit and Wolf, 2003). A more recent approach has been to model assets using *market graphs* (Boginski *et al.*, 2003), that is, based on graph-theoretic techniques. Intuitively, a universe of assets can be naturally modelled as a network of vertices on a graph, whereby an edge between two vertices (assets) designates both the existence of a link and the degree of similarity between assets (Simon, 1991).

Remark 27: A graph-theoretic perspective offers an interpretable explanation for the underperformance of minimum-variance optimization (MVO) techniques in practice. Namely, since the covariance matrix $\mathbf{\Sigma}$ is dense, standard multivariate models implicitly assume full connectivity of the graph, and are therefore not adequate to account for the structure inherent to real-world markets (Calkin and Lopez de Prado, 2014a,b, 2016). *Moreover, it can be shown that the optimal holdings under the MVO framework are inversely proportional to the vertex centrality, thereby suggesting over-investing in assets with low centrality* (Li *et al.*, 2019; Peralta and Zareei, 2016).

Intuitively, it would be highly desirable to remove unnecessary graph edges in order to more appropriately model the underlying structure between assets (graph vertices); this can be achieved through *vertex clustering* of the market graph (Boginski *et al.*, 2003). Various portfolio diversification frameworks employ this technique to allocate capital within and across clusters of assets at multiple hierarchical levels. For instance, the *hierarchical risk parity* scheme (Calkin and Lopez de Prado, 2016) employs an inverse-variance weighting allocation which is based on the number of assets within each asset cluster. Similarly, the *hierarchical clustering based asset allocation* in Raffinot (2017) finds a diversified weighting by distributing capital equally among each of the cluster hierarchies.

Despite mathematical elegance and physical intuition, direct vertex clustering is an NP hard problem. Consequently, existing graph-theoretic portfolio constructions employ combinatorial optimization formulations (Boginski *et al.*, 2003, 2005, 2006, 2014; Gunawardena *et al.*, 2012, Kalyagin *et al.*, 2014), which become computationally intractable for large graph systems. To alleviate this issue, we employ the *minimum cut* vertex clustering method to the graph of portfolio assets, to introduce the concept of *portfolio cut* (Scalzo *et al.*, 2020). In this way, smaller graph partitions (cuts) can be evaluated quasi-optimally, using algebraic methods, and in an efficient and rigorous manner.

13.1 Structure of Market Graph

A universe of N assets can be represented as a set of vertices on a *market graph* (Boginski *et al.*, 2003), whereby the edge weight, W_{mn}, between vertices m and n is defined as the absolute correlation coefficient, $|\rho_{mn}|$, of their respective returns of assets m and n, that is

$$W_{mn} = \frac{|\sigma_{mn}|}{\sqrt{\sigma_{mm}\sigma_{nn}}} = |\rho_{mn}| \tag{13.4}$$

where $\sigma_{mn} = \text{cov}\{r_t(m), r_t(n)\}$ is the covariance of returns between the assets m and n. In this way, we have $W_{mn} = 0$ if the assets m and n are statistically independent (not connected), and $W_{mn} > 0$ if they are statistically dependent (connected on a graph). Note that the resulting weight matrix is symmetric, $\mathbf{W}^T = \mathbf{W}$.

13.2 Minimum Cut Based Vertex Clustering

Vertex clustering aims to group together vertices from the asset universe, $\mathcal{V}$, into multiple disjoint *clusters*, $\mathcal{V}_i$. For a market graph, assets which are grouped into a cluster, $\mathcal{V}_i$, are expected to exhibit a larger degree of mutual within-cluster statistical dependency than with the assets in other clusters, $\mathcal{V}_j$, $j \neq i$. The most popular classical graph cut methods are based on finding the minimum set of edges whose removal would disconnect a graph in some "optimal" sense; this is referred to as *minimum cut* based clustering (Schaeffer, 2007) (see Part I of this monograph for a comprehensive review of the minimum graph cut problem and other graph spectral clustering methods).

Consider an N-vertex market graph, $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, which is grouped into $K = 2$ disjoint subsets of vertices, $\mathcal{V}_1 \subset \mathcal{V}$ and $\mathcal{V}_2 \subset \mathcal{V}$, with $\mathcal{V}_1 \cup \mathcal{V}_2 = \mathcal{V}$ and $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$. A cut of this graph, for the given clusters, $\mathcal{V}_1$ and $\mathcal{V}_2$, is equal to a sum of all weights that correspond to the edges which connect the vertices between the subsets, $\mathcal{V}_1$ and $\mathcal{V}_2$, that is

$$Cut(\mathcal{V}_1, \mathcal{V}_2) = \sum_{m \in \mathcal{V}_1} \sum_{n \in \mathcal{V}_2} W_{mn}. \tag{13.5}$$

A cut which exhibits the minimum value of the sum of weights between the disjoint subsets, $\mathcal{V}_1$ and $\mathcal{V}_2$, considering all possible divisions of the set of vertices, $\mathcal{V}$, is referred to as *the minimum cut*.

Finding the minimum cut in (13.5) is a relatively easy problem and can be solved efficiently (Stoer and Wagner, 1997). However, in practice, this minimum cut formation in (13.5) often leads to unsatisfactory performance (Von Luxburg, 2007). For example, assume that all the weights are positive and that we allow an empty set as a cluster; upon taking one cluster as an empty set and another cluster as a whole graph, that would yield the minimum cut, which is 0. This result is not a reasonable partition we desire. To overcome this problem, we may "balance" the sizes of cluster and cut, i.e., each cluster should be reasonably large, while at the same time the cut itself should be minimized. Instead of using (13.5), two balanced cuts are often used, *Ratio Cut* (Hagen and Kahng, 1992) and *Normalized Cut* (Shi and Malik, 2000), where a balancing term is incorporated into the cut in (13.5).

Within graph cuts, a number of optimization approaches may be employed to enforce some desired properties on graph clusters:

(i) *Ratio cut.* The value of $Cut(\mathcal{V}_1, \mathcal{V}_2)$ is normalized by an additional term to enforce the subsets, $\mathcal{V}_1$ and $\mathcal{V}_2$, to be *simultaneously as large as possible.* The ratio cut formulation is given by Hagen and Kahng (1992)

$$CutR(\mathcal{V}_1, \mathcal{V}_2) = \Big(\frac{1}{N_1} + \frac{1}{N_2}\Big) \sum_{m \in \mathcal{V}_1} \sum_{n \in \mathcal{V}_2} W_{mn} \tag{13.6}$$

where N_1 and N_2 are the respective numbers of vertices in the sets $\mathcal{V}_1$ and $\mathcal{V}_2$. Since $N_1 + N_2 = N$, the term $\frac{1}{N_1} + \frac{1}{N_2}$ reaches its minimum for $N_1 = N_2 = \frac{N}{2}$.

(ii) *Volume normalized cut.* Since the vertex weights are involved when designing the size of subsets $\mathcal{V}_1$ and $\mathcal{V}_2$, then by defining *the volumes* of these sets as $V_1 = \sum_{n \in \mathcal{V}_1} D_{nn}$ and $V_2 = \sum_{n \in \mathcal{V}_2} D_{nn}$, we arrive at the volume normalized cut (Shi and Malik, 2000) (see also Part I)

$$CutN(\mathcal{V}_1, \mathcal{V}_2) = \Big(\frac{1}{V_1} + \frac{1}{V_2}\Big) \sum_{m \in \mathcal{V}_1} \sum_{n \in \mathcal{V}_2} W_{mn}. \tag{13.7}$$

Since $V_1 + V_2 = V$, the term $\frac{1}{V_1} + \frac{1}{V_2}$ reaches its minimum for $V_1 = V_2 = \frac{V}{2}$. Notice that vertices with a higher degree, D_{nn}, are considered as structurally more important than those with lower degrees. In turn, for market graphs, assets with a higher average statistical dependence to other assets are considered as more *central.*

Remark 28: It is important to note that clustering results based on the two above graph cut forms are different. While the ratio cut in (i) favors the clustering into subsets with (almost) equal number of vertices, the volume normalized cut in (ii) favors subsets with (almost) equal volumes, that is, subgraphs with vertices exhibiting (almost) equal average statistical dependence to the other vertices.

Remark 29: Although the optimization algorithm for the cut in (13.5) is simple, by introducing the balancing terms into this cut, the task

of finding the minimum of the objective functions in (13.6) and (13.7) becomes NP hard (Von Luxburg, 2007; Wagner and Wagner, 1993). However, if we relax the problem from a discrete valued to a real valued one, then this boils down to the eigenproblem of graph Laplacian, which is considered next.

13.3 Spectral Bisection Based Minimum Cut

To overcome the computational burden of finding the ratio cut, we may opt for an approximative spectral solution which clusters vertices using the eigenvectors of the graph Laplacian, $\mathbf{L}$. The algorithm employs the second (Fiedler, 1973) eigenvector of the graph Laplacian, $\mathbf{u}_2 \in \mathbb{R}^N$, to yield *quasi-optimal* vertex clustering on a graph. Despite its simplicity, the algorithm is typically accurate and gives a good approximation to the minimum cut (Ng *et al.*, 2002; Spielman and Teng, 2007).

To relate the problem of the minimum cut in (13.6) and (13.7) to that of eigenanalysis of graph Laplacian, we employ an *indicator vector*, denoted by $\mathbf{x} \in \mathbb{R}^N$ (Stanković *et al.*, 2019a), for which the elements take sub-graph-wise constant values within each disjoint subset (cluster) of vertices, with these constants taking different values for different clusters of vertices. In other words, the elements of $\mathbf{x}$ uniquely reflect the assumed cut of the graph into disjoint subsets $\mathcal{V}_1, \mathcal{V}_2 \subset \mathcal{V}$.

For a general graph, we consider two possible solutions for the indicator vector, $\mathbf{x}$, that satisfy the subset-wise constant form:

(i) *Ratio cut.* It can be shown that if the indicator vector is defined as (see Part I of this monograph)

$$x(n) = \begin{cases} \dfrac{1}{N_1}, & \text{for } n \in \mathcal{V}_1, \\ -\dfrac{1}{N_2}, & \text{for } n \in \mathcal{V}_2, \end{cases} \tag{13.8}$$

then the ratio cut, $CutR(\mathcal{V}_1, \mathcal{V}_2)$ in (13.6), is equal to the Rayleigh quotient of $\mathbf{L}$ and $\mathbf{x}$, that is

$$CutR(\mathcal{V}_1, \mathcal{V}_2) = \frac{\mathbf{x}^T \mathbf{L} \mathbf{x}}{\mathbf{x}^T \mathbf{x}}. \tag{13.9}$$

Therefore, the indicator vector, $\mathbf{x}$, which minimizes the ratio cut also minimizes (13.9). From the indicator vector, we see

$$\sum_{n\in V} x(n) = \sum_{n\in V_1} x(n) + \sum_{n\in V2} x(n) = N_1 \times \frac{1}{N_1} - N_2 \times \frac{1}{N_2} = 0. \tag{13.10}$$

In other words, we can say that the vector $\mathbf{x}$ is orthogonal to $\mathbf{1}$. Moreover, we can see that the objective function in (13.9) is invariant of the scale of $\mathbf{x}$. From this discussion, we can relax the problem of the objective function in (13.6) through the constraints, as

$$\min_{\mathbf{x}} \mathbf{x}^T\mathbf{L}\mathbf{x}, \quad \text{subject to } \mathbf{x}^T\mathbf{x} = 1, \quad \text{and} \quad \mathbf{x}^T\mathbf{1} = 0. \tag{13.11}$$

Given that the considered graph is undirected and therefore $\mathbf{L}$ is symmetric, the first eigenvector of the graph Laplacian is constant (proportional to vector $\mathbf{1}$, $\mathbf{u}_0 = \mathbf{1}/\sqrt{N}$) and the associated first eigenvalue is $\lambda_0 = 0$. Therefore, by the Rayleigh–Ritz theorem, the solution to the objective function in (13.11) is given by the second eigenvector of the graph Laplacian, $\mathbf{L}$, obtained as

$$\mathbf{L}\mathbf{x} = \lambda_1\mathbf{x}, \tag{13.12}$$

with the second eigenvalue, $\lambda_k = \lambda_1$.

(ii) Volume normalized cut. Similarly, by defining $\mathbf{x}$ as

$$x(n) = \begin{cases} \dfrac{1}{V_1}, & \text{for } n \in \mathcal{V}_1, \\ -\dfrac{1}{V_2}, & \text{for } n \in \mathcal{V}_2, \end{cases} \tag{13.13}$$

the volume normalized cut, $CutN(\mathcal{V}_1, \mathcal{V}_2)$ in (13.7), takes the form of a generalized Rayleigh quotient of $\mathbf{L}$, given by (see again Part I)

$$CutN(\mathcal{V}_1, \mathcal{V}_2) = \frac{\mathbf{x}^T\mathbf{L}\mathbf{x}}{\mathbf{x}^T\mathbf{D}\mathbf{x}}. \tag{13.14}$$

Similarly to the ratio cut, we see from the indicator vector that

$$\begin{aligned}\sum_{n\in V} d(n)x(n) &= \sum_{n\in V_1} d(n)x(n) + \sum_{n\in V2} d(n)x(n) \\ &= \sum_{n\in V_1} d(n) \times \frac{1}{V_1} - \sum_{n\in V_2} d(n) \times \frac{1}{V_2} \\ &= V_1 \times \frac{1}{V_1} - V_2 \times \frac{1}{V_2} = 0, \end{aligned} \tag{13.15}$$

which yields $(\mathbf{Dx})^T\mathbf{1} = 0$. Also, the objective function is invariant to the scale of $\mathbf{x}$. Therefore, we can formulate the optimization problem from the objective function (13.14) as

$$\min_{\mathbf{x}} \mathbf{x}^T\mathbf{Lx}, \quad \text{subject to } \mathbf{x}^T\mathbf{Dx} = 1, \quad \text{and} \quad (\mathbf{Dx})^T\mathbf{1} = 0. \tag{13.16}$$

The solution is given by the second generalized eigenvector of the generalized eigenproblem of the graph Laplacian as

$$\mathbf{Lx} = \lambda_1\mathbf{Dx}, \tag{13.17}$$

since $\mathbf{D}^{-1/2}\mathbf{1}$ is the first generalized eigenvector of graph Laplacian.

For the spectral solutions above, the membership of a vertex, n, to either the subset $\mathcal{V}_1$ or $\mathcal{V}_2$ is uniquely defined by the *sign* of the indicator vector, $\mathbf{x} = \mathbf{u}_1$, that is

$$\text{sign}(x(n)) = \begin{cases} 1, & \text{for } n \in \mathcal{V}_1, \\ -1, & \text{for } n \in \mathcal{V}_2. \end{cases} \tag{13.18}$$

Notice that a scaling of $\mathbf{x}$ by any constant would not influence the solution for clustering into the subsets $\mathcal{V}_1$ or $\mathcal{V}_2$.

13.4 Repeated Portfolio Cuts

Although the above analysis has focused on the case with $K = 2$ disjoint sub-graphs, it can be straightforwardly generalized to $K \geq 2$ disjoint sub-graphs through the method of *repeated bisection*.

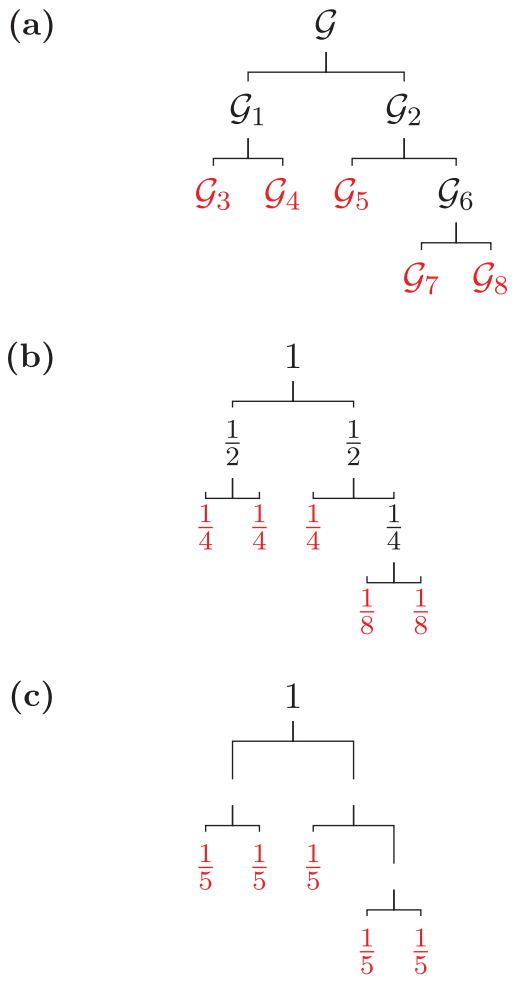


Figure 13.1: Graph cut based asset allocation strategies. (a) Hierarchical graph structure resulting from $K = 4$ portfolio cuts. (b) A graph tree based on the $\frac{1}{2^{K_i}}$ scheme. (c) A graph tree based on the $\frac{1}{K+1}$ scheme.

A single application of the portfolio cut on the market graph, $\mathcal{G}$, produces two disjoint sub-graphs, $\mathcal{G}_1$ and $\mathcal{G}_2$, as illustrated in Figure 13.1(a). Notice that in this way we construct a hierarchical binary tree structure, whereby the union of the *leaves* of the network is equal to the original market graph, $\mathcal{G}$. We can then perform a subsequent portfolio cut operation on one or both of the leaves based on some suitable criterion (e.g., the leaf with the greatest number of vertices or volume). Therefore, $(K + 1)$ disjoint sub-graphs (leaves) can be obtained by performing the portfolio cut procedure K times (Scalzo *et al.*, 2020).

Example 41: Figure 13.1(a) illustrates the hierarchical structure resulting from $K = 4$ portfolio cuts of a market graph, $\mathcal{G}$. The leaves

of the resulting binary tree are denoted by $\{\mathcal{G}_3, \mathcal{G}_4, \mathcal{G}_5, \mathcal{G}_7, \mathcal{G}_8\}$ (in red), whereby the number of disjoint sub-graphs is equal to $(K+1) = 5$. Notice that the union of the leaves amounts to the original graph, i.e., $\mathcal{G}_3 \cup \mathcal{G}_4 \cup \mathcal{G}_5 \cup \mathcal{G}_7 \cup \mathcal{G}_8 = \mathcal{G}$.

13.5 Graph Asset Allocation Schemes

We next elaborate upon some intuitive asset allocation strategies, inspired by the work in Calkin and Lopez de Prado (2016) and Raffinot (2017), which naturally builds upon the portfolio cut. The aim is to determine a diversified weighting scheme by distributing capital among the disjoint clusters (leaves) so that highly correlated assets within a given cluster receive the same total allocation, thereby being treated as a single investment entity.

Upon denoting the portion of the total capital allocated to a cluster $\mathcal{G}_i$ by w_i, we consider two simple asset allocation schemes:

(AS1) $w_i = \frac{1}{2^{K_i}}$, where K_i is the number of portfolio cuts required to obtain a sub-graph $\mathcal{G}_i$;

(AS2) $w_i = \frac{1}{K+1}$, where $(K+1)$ is the number of disjoint sub-graphs.

Remark 30: An equally-weighted asset allocation strategy may now be employed within each cluster, i.e., every asset within the ith cluster, $\mathcal{G}_i$, will receive a weighting equal to $\frac{w_i}{N_i}$.

Remark 31: The weighting scheme in AS1 above is closely related to the strategy proposed in Raffinot (2017), while the scheme in AS2 is inspired by the generic equal-weighted (EW) allocation scheme (De Miguel *et al.*, 2009). These schemes are convenient in that they require no assumptions regarding the across-cluster statistical dependence. In addition, unlike the EW scheme, they implicitly consider the inherent market risks (asset correlation) by virtue of the portfolio cut formulation, which is based on the eigenanalysis of the market graph Laplacian, $\mathbf{L}$.

Example 42: Figures 13.1(b) and (c) illustrate respectively the asset allocation schemes in AS1 and AS2 for $K = 4$ portfolio cuts, based on

the market graph partitioning in Figure 13.1(a). Notice that the weights associated to the disjoint sub-graphs (leaves in red) sum up to unity.

13.6 Numerical Example

The performance of the portfolio cuts and the associated graph-theoretic asset allocation schemes was investigated using historical price data comprising of the 100 most liquid stocks in the S&P 500 index, based on the average trading volume, in the period 2014-01-01 to 2018-01-01. The data was split into: (i) the *in-sample* dataset (2014-01-01 to 2015-12-31) which was used to estimate the asset correlation matrix and to compute the portfolio cuts; and (ii) the *out-sample* dataset (2016-01-01 to 2018-01-01), used to objectively quantify the profitability of the asset allocation strategies (Scalzo *et al.*, 2020).

Figure 13.2 displays the Kth iterations, for $K = 1, 2, 10$, of the normalized portfolio cut in (13.9), applied to the original 100-vertex

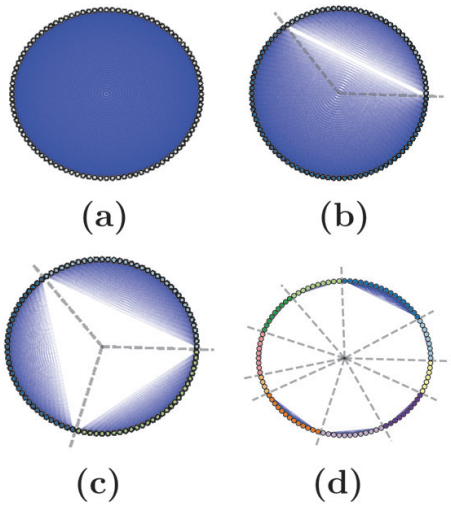
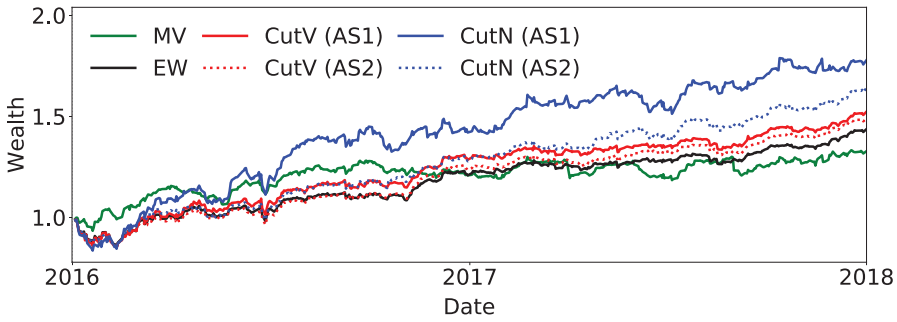


Figure 13.2: Visualization of the 100-vertex market graph connectivity for the 100 most liquid stocks in S&P 500 index, and its partitions into disjoint sub-graphs (separated by dashed grey lines). The edges (blue lines) were calculated based on the correlation between assets. (a) Fully connected market graph with 5050 edges. (b) Partitioned graph after $K = 1$ portfolio cuts (CutN), with 2746 edges. (c) Partitioned graph after $K = 2$ portfolio cuts (CutN), with 1731 edges. (d) Partitioned graph after $K = 10$ portfolio cuts (CutN), with 575 edges. Notice that the number of edges required to model the market graph is significantly reduced with each subsequent portfolio cut, since $\sum_{i=1}^{K+1} \frac{1}{2}(N_i^2 + N_i) < \frac{1}{2}(N^2 + N)$, $\forall K > 0$.



(a) Evolution of wealth for both the traditional (EW and MV) and graph-theoretic asset allocation strategies, based on ($K = 10$) portfolio cuts.

Cut Method	Allocation	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$	$K=10$
CutN	AS1	1.82	1.80	1.80	1.93	1.96	**1.98**
CutN	AS2	1.82	1.81	1.94	2.03	1.95	**2.05**
CutR	AS1	1.93	2.01	2.08	2.23	2.22	**2.25**
CutR	AS2	1.93	2.04	2.17	**2.65**	2.51	2.48

(b) Sharpe ratios attained for a varying number of portfolio cuts K.

Figure 13.3: Out-sample performance of the graph cut based asset allocation strategies. Notice that the Sharpe ratio typically improves with each subsequent portfolio cut. The traditional portfolio strategies, EW and MV, attained the respective Sharpe ratios of $\text{SR}_{\text{EW}} = 1.85$ and $\text{SR}_{\text{MV}} = 1.6$.

market graph obtained from the in-sample data set. Next, for the out-sample dataset, graph representations of the portfolio, for the number of cuts K varying in the range $[1, 10]$, were employed to assess the performance of the asset allocation schemes described in Section 13.5. The standard equally-weighted (EW) and minimum-variance (MV) portfolios were also simulated for comparison purposes, with the results displayed in Figure 13.3.

Conforming with the findings in Calkin and Lopez de Prado (2016) and Raffinot (2017), the proposed graph asset allocation schemes consistently delivered lower out-sample variance than the standard EW and MV portfolios, thereby attaining a higher *Sharpe ratio*, i.e., the ratio of the mean to the standard deviation of portfolio returns. This verifies that the removal of possibly spurious statistical dependencies in the "raw" format, through portfolio cuts, allows for robust and flexible portfolio constructions.

Such an approach enables the creation of graph-theoretic capital allocation schemes, based on measures of connectivity which are inherent to the portfolio cut formulation. In addition, the proposed portfolio construction employs full information contained in the asset covariance matrix, and without requiring its inversion, even in the critical cases of limited data length or singular covariance matrices.

14

Conclusion

In many modern applications, graph topology is not known a priori and hence its determination becomes part of the problem definition, rather than serving as prior knowledge to aid solution. To perform simultaneous estimation of both data on a graph and the underlying graph topology, without loss of generality we assume that the vertices (their number, location, etc.) are given, while the edges and their associated weights form part of the solution to the problem under consideration. Three possible scenarios for the estimation of graph edges from the data observed on a graph have been considered. Namely, in various sensor network sensing setups (temperature, pressure, transportation) the locations of the sensor positions (vertices) may be known while the vertex distances convey physical meaning about data and inter-sensor dependence and thus may be employed for weight determination. Another possibility is to employ the covariance and precision matrices, which are commonly used as data similarity metrics and are thus a natural choice of a metric for learning graph topology from data. The third scenario are graphs for which the relations among the sensing positions are physically well defined, such as in electric circuits, power

networks, linear heat transfer, social and computer networks, and spring-mass systems. Next, the problem of simulation of graph signals has been addressed and a detailed derivation and elaboration of sparsity structure promoting optimization approaches, such as the LASSO and graph-based version of LASSO (GLASSO), has been given. The inherent connection between graphs and deep neural networks (DNNs) has been further addressed, and the concepts of graph neural networks (GNN) and graph convolutional neural networks (GCN) have been introduced. It has been shown that the diffusion process on graphs underpins the operation of GNNs. The enormous potential of the combination of the universal function approximation property of neural networks with the elegance and generality of graph models has been demonstrated through the concepts of recurrent GNNs, spatial GNNs, spectral GNNs, together with the interpretation of graph signal filtering as a diffusion process in a "neural network" language. The advantages of these concepts have been illustrated over the paradigms of semi-supervised learning and label propagation, while the use of GNNs in graph link prediction has been addressed based on an innovative but natural combination of characteristic functions and generative adversarial nets, referred to as reciprocal adversarial learning via characteristic functions (RCF-GAN). Furthermore, the application of graphs in Big Data scenarios has been demonstrated through their link with tensors, and tensor factorizations. This is particularly significant, as multidimensional graphs are common in practice, but are inadequately modelled through their imbalanced and "flat view" adjacency matrices. To this end, we show that multilinear algebra, whereby multidimensional graphs are modelled via the corresponding adjacency tensor, is a natural choice to discover intrinsic relations in such multidimensional data. This has led to the concept of multi-graph tensor network (MGNT), which serves as a general framework for neural network learning in big data settings and on multiple irregular domains. Finally, innovative and comprehensively elaborated case studies have been given in support of the concepts, ranging from portfolio cuts in finance to the modelling of vulnerability of stations in underground metro traffic.

Acknowledgments

We wish to express our sincere gratitude to Yao Lei Xu, Kriton Konstantinidis, Ghena Hammour, Shota Saito, and Giacomo Kahn whose thorough proofreading and deep insight have been of great help at various stages of manuscript preparation.

References

Atwood, J. and D. Towsley (2016). "Diffusion-convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 1993–2001.

Baba, K., R. Shibata, and M. Sibuya (2004). "Partial correlation and conditional correlation as measures of conditional independence". *Australian & New Zealand Journal of Statistics*. 46(4): 657–664.

Bacciu, D. and L. Di Sotto (2019). "A non-negative factorization approach to node pooling in graph convolutional neural networks". In: *Proceedings of the International Conference of the Italian Association for Artificial Intelligence*. Springer. 294–306.

Bacciu, D., F. Errica, and A. Micheli (2018). "Contextual graph Markov model: A deep and generative approach to graph processing". *arXiv preprint arXiv:1805.10636*.

Bacciu, D. and D. P. Mandic (2020). "Tensor decompositions in deep learning". In: *Proc. of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN'20)*. 441–450.

Baingana, B. and G. B. Giannakis (2016). "Tracking switched dynamic network topologies from information cascades". *IEEE Transactions on Signal Processing*. 65(4): 985–997.

Banerjee, O., L. E. Ghaoui, and A. d'Aspremont (2008). "Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data". *Journal of Machine Learning Research*. 9(Mar): 485–516.

Belkin, M. and P. Niyogi (2003). "Laplacian eigenmaps for dimensionality reduction and data representation". *Neural Computation*. 15(6): 1373–1396.

Belkin, M. and P. Niyogi (2008). "Towards a theoretical foundation for Laplacian-based manifold methods". *Journal of Computer and System Sciences*. 74(8): 1289–1308.

Berge, C. (1984). *Hypergraphs: Combinatorics of Finite Sets*. Vol. 45. Elsevier.

Black, F. and R. Litterman (1992). "Global portfolio optimization". *Financial Analysts Journal*. 48(5): 280–291.

Boginski, V., S. Butenko, and P. M. Pardalos (2003). "On structural properties of the market graph". In: *Innovations in Financial and Economic Networks*. Ed. by A. Nagurney. Edward Elgar Publishers. 29–45.

Boginski, V., S. Butenko, and P. M. Pardalos (2005). "Statistical analysis of financial networks". *Computational Statistics & Data Analysis*. 48(2): 431–443.

Boginski, V., S. Butenko, and P. M. Pardalos (2006). "Mining market data: A network approach". *Computers & Operations Research*. 33(11): 3171–3184.

Boginski, V., S. Butenko, S. O., S. Trunkhanov, and J. Gil Lafuente (2014). "A network-based data mining approach to portfolio selection via weighted clique relaxations". *Annals of Operations Research*. 216: 23–34.

Bohannon, A. W., B. M. Sadler, and R. V. Balan (2019). "A filtering framework for time-varying graph signals". In: *Vertex-Frequency Analysis of Graph Signals*. Ed. by L. Stanković and E. Sejdić. Springer. 341–376.

Bojchevski, A., O. Shchur, D. Zügner, and S. Günnemann (2018). "NetGAN: Generating graphs via random walks". *arXiv preprint arXiv:1803.00816*.

Brandes, U. (2005). *Network Analysis: Methodological Foundations*. Springer.

Bruna, J., W. Zaremba, A. Szlam, and Y. LeCun (2013). "Spectral networks and locally connected networks on graphs". *arXiv preprint arXiv:1312.6203*.

Caetano, T. S., J. J. McAuley, L. Cheng, Q. V. Le, and A. J. Smola (2009). "Learning graph matching". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 31(6): 1048–1058.

Calkin, N. J. and M. Lopez de Prado (2014a). "Stochastic flow diagrams". *Algorithmic Finance*. 3(1–2): 21–42.

Calkin, N. J. and M. Lopez de Prado (2014b). "The topology of macro financial flows: An application of stochastic flow diagrams". *Algorithmic Finance*. 3(1): 43–85.

Calkin, N. J. and M. Lopez de Prado (2016). "Building diversified portfolios that outperform out of sample". *The Journal of Portfolio Management*. 42(4): 59–69.

Calvi, G. G., A. Moniri, M. Mahfouz, Q. Zhao, and D. P. Mandic (2019). "Compression and interpretability of deep neural networks via Tucker tensor layer: From first principles to tensor valued back-propagation". *arXiv preprint arXiv:1903.06133*.

Camponogara, E. and L. F. Nazari (2015). "Models and algorithms for optimal piecewise-linear function approximation". *Mathematical Problems in Engineering*. 2015.

Candès, E. J., J. Romberg, and T. Tao (2006). "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information". *IEEE Transactions on Information Theory*. 52(2): 489–509.

Chami, I., S. Abu-El-Haija, B. Perozzi, C. Re, and K. Murphy (2020). "Machine learning on graphs: A model and comprehensive taxonomy". *arXiv preprint arXiv:2005.03675*.

Chen, G., D. R. Glen, Z. S. Saad, J. P. Hamilton, M. E. Thomason, I. H. Gotlib, and R. W. Cox (2011). "Vector autoregression, structural equation modeling, and their synthesis in neuroimaging data analysis". *Computers in Biology and Medicine*. 41(12): 1142–1155.

Chen, S., A. Sandryhaila, J. M. Moura, and J. Kovačević (2015). "Signal recovery on graphs: Variation minimization". *IEEE Transactions on Signal Processing*. 63(17): 4609–4624.

Chen, S., R. Varma, A. Singh, and J. Kovačević (2016). "Signal recovery on graphs: Fundamental limits of sampling strategies". *IEEE Transactions on Signal and Information Processing Over Networks*. 2(4): 539–554.

Chepuri, S. P., S. Liu, G. Leus, and A. O. Hero (2017). "Learning sparse graphs under smoothness prior". In: *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 6508–6512.

Cichocki, A., N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic (2016). "Tensor networks for dimensionality reduction and large-scale optimization. Part 1: Low-rank tensor decompositions". *Foundations and Trends® in Machine Learning*. 9(4–5): 249–429.

Cichocki, A., A.-H. Phan, Q. Zhao, N. Lee, I. V. Oseledets, M. Sugiyama, and D. Mandic (2017). "Tensor networks for dimensionality reduction and large-scale optimization. Part 2: Applications and future perspectives". *Foundations and Trends® in Machine Learning*. 9(6): 431–673.

Cioacă, T., B. Dumitrescu, and M.-S. Stupariu (2019). "Graph-based wavelet multiresolution modeling of multivariate terrain data". In: *Vertex-Frequency Analysis of Graph Signals*. Ed. by L. Stanković and E. Sejdić. Springer. 479–507.

Clarke, R., H. De Silva, and S. Thorley (2002). "Portfolio constraints and the fundamental law of active management". *Financial Analysts Journal*. 58: 48–66.

Cooper, J. and A. Dutle (2012). "Spectra of uniform hypergraphs". *Linear Algebra and Its Applications*. 436(9): 3268–3292.

Dai, H., Z. Kozareva, B. Dai, A. Smola, and L. Song (2018). "Learning steady-states of iterative algorithms over graphs". In: *Proceedings of the International Conference on Machine Learning*. 1114–1122.

Dal Col, A., P. Valdivia, F. Petronetto, F. Dias, C. T. Silva, and L. G. Nonato (2019). "Wavelet-based visual data exploration". In: *Vertex-Frequency Analysis of Graph Signals*. Ed. by L. Stanković and E. Sejdić. Springer. 459–478.

Das, A., A. L. Sampson, C. Lainscsek, L. Muller, W. Lin, J. C. Doyle, S. S. Cash, E. Halgren, and T. J. Sejnowski (2017). "Interpretation of the precision matrix and its application in estimating sparse brain connectivity during sleep spindles from human electrocorticography recordings". *Neural Computation*. 29(3): 603–642.

De Cao, N. and T. Kipf (2018). "MolGAN: An implicit generative model for small molecular graphs". *arXiv preprint arXiv:1805.11973*.

De Miguel, V., L. Garlappi, and R. R. Uppal (2009). "Optimal versus naive diversification: How inefficient is the $1/N$ portfolio strategy?" *Review of Financial Studies*. 22: 1915–1953.

Dees, B. S., A. G. Constantinides, and D. P. Mandic (2019). "Graph theory and metro traffic modelling". *arXiv preprint arXiv:1912.05964*.

Defferrard, M., X. Bresson, and P. Vandergheynst (2016). "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Advances in Neural Information Processing Systems*. 3844–3852.

Dempster, A. P. (1972). "Covariance selection". *Biometrics*: 157–175.

Doersch, C. (2016). "Tutorial on variational autoencoders". *arXiv preprint arXiv:1606.05908*.

Dong, X., D. Thanou, P. Frossard, and P. Vandergheynst (2015). "Learning graphs from signal observations under smoothness prior". June [online]. Available: URL: http://arXiv.org/abs/1406.7842.

Dong, X., D. Thanou, P. Frossard, and P. Vandergheynst (2016). "Learning Laplacian matrix in smooth graph signal representations". *IEEE Transactions on Signal Processing*. 64(23): 6160–6173.

Dong, X., D. Thanou, M. Rabbat, and P. Frossard (2019). "Learning graphs from data: A signal representation perspective". *IEEE Signal Processing Magazine*. 36(3): 44–63.

Dorfler, F. and F. Bullo (2012). "Kron reduction of graphs with applications to electrical networks". *IEEE Transactions on Circuits and Systems I: Regular Papers*. 60(1): 150–163.

Epskamp, S. and E. I. Fried (2018). "A tutorial on regularized partial correlation networks". *Psychological Methods*. 23(4): 617.

Fiedler, M. (1973). "Algebraic connectivity of graphs". *Czechoslovak Mathematical Journal*. 23(2): 298–305.

Freeman, L. C. (1977). "A set of measures of centrality based on betweenness". *Sociometry*. 40: 35–41.

Friedman, J., T. Hastie, and R. Tibshirani (2008). "Sparse inverse covariance estimation with the graphical LASSO". *Biostatistics*. 9(3): 432–441.

Gauvin, L., A. Panisson, and C. Cattuto (2014). "Detecting the community structure and activity patterns of temporal networks: A non-negative tensor factorization approach". *PLOS ONE*. 9(1): e86028.

Giannakis, G. B., Y. Shen, and G. V. Karanikolas (2018). "Topology identification and learning over graphs: Accounting for nonlinearities and dynamics". *Proceedings of the IEEE*. 106(5): 787–807.

Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl (2017). "Neural message passing for quantum chemistry". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. 1263–1272.

Gori, M., G. Monfardini, and F. Scarselli (2005). "A new model for learning in graph domains". In: *Proceedings of the IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. 729–734.

Grotas, S., Y. Yakoby, I. Gera, and T. Routtenberg (2019). "Power systems topology and state estimation by graph blind source separation". *IEEE Transactions on Signal Processing*. 67(8): 2036–2051.

Grover, A. and J. Leskovec (2016). "Node2Vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 855–864.

Grover, A., A. Zweig, and S. Ermon (2019). "Graphite: Iterative generative modeling of graphs". In: *Proc. International Conference on Machine Learning*. 2434–2444.

Gu, Y. and X. Wang (2019). "Local-set-based graph signal sampling and reconstruction". In: *Vertex-Frequency Analysis of Graph Signals*. Ed. by L. Stanković and E. Sejdić. Springer. 255–292.

Gunawardena, A. A., R. R. Meyer, and W. L. Dougan (2012). "Optimal selection of an independent set of cliques in a market graph". In: *Proceedings of the International Conference on Economics, Business and Marketing Management*. 281–285.

Hagen, L. and A. B. Kahng (1992). "New spectral methods for ratio cut partitioning and clustering". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 11(9): 1074–1085.

Hamilton, W., Z. Ying, and J. Leskovec (2017). "Inductive representation learning on large graphs". In: *Advances in Neural Information Processing Systems*. 1024–1034.

Hammond, D. K., P. Vandergheynst, and R. Gribonval (2011). "Wavelets on graphs via spectral graph theory". *Applied and Computational Harmonic Analysis*. 30(2): 129–150.

Hamon, R., P. Borgnat, P. Flandrin, and C. Robardet (2019). "Transformation from graphs to signals and back". In: *Vertex-Frequency Analysis of Graph Signals*. Ed. by L. Stanković and E. Sejdić. Springer. 111–139.

Hasanzadeh, A., E. Hajiramezanali, K. Narayanan, N. Duffield, M. Zhou, and X. Qian (2019). "Semi-implicit graph variational auto-encoders". In: *Advances in Neural Information Processing Systems*. 10712–10723.

Ioannidis, V. N., D. Berberidis, and G. B. Giannakis (2019a). "GraphSAC: Detecting anomalies in large-scale graphs". *arXiv preprint arXiv:1910.09589*.

Ioannidis, V. N., Y. Shen, and G. B. Giannakis (2019b). "Semi-blind inference of topologies and dynamical processes over dynamic graphs". *IEEE Transactions on Signal Processing*. 67(9): 2263–2274.

Jeh, G. and J. Widom (2002). "SIMRANK: A measure of structural-context similarity". In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 538–543.

Kalofolias, V. (2016). "How to learn a graph from smooth signals". In: *Proceedings of the Artificial Intelligence and Statistics*. 920–929.

Kalyagin, V., A. Koldanov, P. Koldanov, and V. Zamaraev (2014). "Market graph and Markowitz model". In: *Optimization in Science and Engineering*. Ed. by T. M. Rassias, C. A. Floudas, and S. Butenko. Springer. 293–306.

Kaplan, D. (2008). *Structural Equation Modeling: Foundations and Extensions*. Vol. 10. Sage Publications.

Katsimpras, G. and G. Paliouras (2020). "Class-aware tensor factorization for multi-relational classification". *Information Processing & Management*. 57(2): 102068.

Kingma, D. P. and M. Welling (2013). "Auto-encoding variational Bayes". *arXiv preprint arXiv:1312.6114*.

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

Kipf, T. N. and M. Welling (2016b). "Variational graph auto-encoders". *arXiv preprint arXiv:1611.07308*.

Kolaczyk, E. D. (2009). *Statistical Analysis of Network Data – Methods and Models*. New York: Springer-Verlag.

Kolm, P. N., R. Tutuncu, and F. J. Fabozzi (2014). "60 years of portfolio optimization: Practical challenges and current trends". *European Journal of Operational Research*. 234(2): 356–371.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". *Proceedings of the IEEE*. 86(11): 2278–2324.

Ledoit, O. and M. Wolf (2003). "Improved estimation of the covariance matrix of stock returns with an application to portfolio selection". *Journal of Empirical Finance*. 10(5): 603–621.

Li, S., Z. Yu, M. Xiang, and D. Mandic (2020). "Reciprocal adversarial learning via characteristic functions". *arXiv preprint arXiv:2006.08413*.

Li, Y., X. F. Jiang, Y. Tian, S. P. Li, and B. Zheng (2019). "Portfolio optimization based on network topology". *Physica A*. 515: 671–681.

Li, Y., D. Tarlow, M. Brockschmidt, and R. Zemel (2015). "Gated graph sequence neural networks". *arXiv preprint arXiv:1511.05493*.

Li, Y., O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia (2018). "Learning deep generative models of graphs". *arXiv preprint arXiv:1803.03324*.

Liben-Nowell, D. and J. Kleinberg (2007). "The link-prediction problem for social networks". *Journal of the American Society for Information Science and Technology*. 58(7): 1019–1031.

Lin, Y. R., Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng (2008). "Facetnet: A framework for analyzing communities and their evolutions in dynamic networks". In: *Proceedings of the International Conference on World Wide Web (WWW)*. 685–694.

Lin, Y., J. Sun, P. Castro, R. Konuru, H. Sundaram, and A. Kelliher (2009). "MetaFac: Community discovery via relational hypergraph factorization". In: *Proceedings of the ACM KDD International Conference on Knowledge Discovery and Data Mining*. 527–536.

Ma, T., J. Chen, and C. Xiao (2018). "Constrained generation of semantically valid graphs via regularizing variational autoencoders". In: *Advances in Neural Information Processing Systems*. 7113–7124.

Mandic, D. (2007). "Machine learning and signal processing applications of fixed point theory". *Tutorial in IEEE ICASSP, 2007*.

Mandic, D. and J. Chambers (2001). *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*. Wiley.

Mandic, D. P. and V. S. L. Goh (2009). *Complex Valued Nonlinear Adaptive Filters: Noncircularity, Widely Linear and Neural Models*. John Wiley & Sons.

Mao, X. and Y. Gu (2019). "Time-varying graph signals reconstruction". In: *Vertex-Frequency Analysis of Graph Signals*. Ed. by L. Stanković and E. Sejdić. Springer. 293–316.

Markowitz, H. (1952). "Portfolio selection". *Journal of Finance*. 7(1): 77–91.

Marques, A., A. Ribeiro, and S. Segarra (2017). "Graph signal processing: Fundamentals and applications to diffusion processes". In: *Proc. of the IEEE Int. Conf. Accoustic, Speech and Signal Processing, (ICASSP), Tutorial, 2017*.

Masuda, N., M. A. Porter, and R. Lambiotte (2017). "Random walks and diffusion on networks". *Physics Reports*. 716: 1–58.

Mateos, G., S. Segarra, A. G. Marques, and A. Ribeiro (2019). "Connecting the dots: Identifying network structure via graph signal processing". *IEEE Signal Processing Magazine*. 36(3): 16–43.

Mei, J. and J. M. Moura (2016). "Signal processing on graphs: Causal modeling of unstructured data". *IEEE Transactions on Signal Processing*. 65(8): 2077–2092.

Meinshausen, N. and P. Bühlmann (2006). "High-dimensional graphs and variable selection with the LASSO". *The Annals of Statistics*. 34(3): 1436–1462.

Micheli, A. (2009). "Neural network for graphs: A contextual constructive approach". *IEEE Transactions on Neural Networks*. 20(3): 498–511.

Monti, F., D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein (2017). "Geometric deep learning on graphs and manifolds using mixture model CNNs". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5115–5124.

Motl, J. and O. Schulte (2015). "The CTU prague relational learning repository". *arXiv preprint arXiv:1511.03086*.

Ng, A. Y., M. I. Jordan, and Y. Weiss (2002). "On spectral clustering: Analysis and an algorithm". *Advances in Neural Information Processing Systems*. 2002: 849–856.

Nickel, M., V. Tresp, and H.-P. Kriegel (2011). "A three-way model for collective learning on multi-relational data". In: *Proceedings of the 28th International Conference on Machine Learning*. 809–816.

Novikov, A., D. Podoprikhin, A. Osokin, and D. P. Vetrov (2015). "Tensorizing neural networks". In: *Advances in Neural Information Processing Systems (NIPS)*. 442–450.

Pan, S., R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang (2018). "Adversarially regularized graph autoencoder for graph embedding". *arXiv preprint arXiv:1802.04407*.

Papalexakis, E. E., L. Akoglu, and D. Lence (2013). "Do more views of a graph help? Community detection and clustering in multi-graphs". In: *Proceedings of the 16th International Conference on Information Fusion*. 899–905.

Pasdeloup, B., V. Gripon, R. Alami, and M. G. Rabbat (2019). "Uncertainty principle on graphs". In: *Vertex-Frequency Analysis of Graph Signals*. Ed. by L. Stanković and E. Sejdić. Springer. 317–340.

Pavez, E. and A. Ortega (2016). "Generalized Laplacian precision matrix estimation for graph signal processing". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2016*. 6350–6354.

Peralta, G. and A. Zareei (2016). "A network approach to portfolio selection". *Journal of Empirical Finance*. 38(A): 157–180.

Perozzi, B., R. Al-Rfou, and S. Skiena (2014). "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 701–710.

Pourahmadi, M. (2011). "Covariance estimation: The GLM and regularization perspectives". *Statistical Science*: 369–387.

Rabiei, H., F. Richard, O. Coulon, and J. Lefèvre (2019). "Estimating the complexity of the cerebral cortex folding with a local shape spectral analysis". In: *Vertex-Frequency Analysis of Graph Signals*. Ed. by L. Stanković and E. Sejdić. Springer. 437–458.

Raffinot, T. (2017). "Hierarchical clustering-based asset allocation". *The Journal of Portfolio Management*. 44(2): 89–99.

Sadhanala, V., Y.-X. Wang, and R. Tibshirani (2016). "Graph sparsification approaches for Laplacian smoothing". In: *Artificial Intelligence and Statistics*. 1250–1259.

Saito, S., D. P. Mandic, and H. Suzuki (2018). "Hypergraph p-Laplacian: A differential geometry view". In: *Proc. of the Thirty-Second AAAI Conference on Artificial Intelligence*. 3984–3991.

Sakiyama, A., Y. Tanaka, T. Tanaka, and A. Ortega (2019). "Eigendecomposition-free sampling set selection for graph signals". *IEEE Transactions on Signal Processing*. 67(10): 2679–2692.

Scalzo, B., L. Stanković, A. G. Constantinides, and D. P. Mandic (2020). "Portfolio cuts: A graph-theoretic framework to diversification". In: *Proc. of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 8454–8458.

Scarselli, F., M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini (2008). "The graph neural network model". *IEEE Transactions on Neural Networks*. 20(1): 61–80.

Schaeffer, S. E. (2007). "Graph clustering". *Computer Science Review*. 1(1): 27–64.

Segarra, S., A. G. Marques, G. Mateos, and A. Ribeiro (2016). "Blind identification of graph filters with multiple sparse inputs." In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 4099–4103.

Segarra, S., A. G. Marques, G. Mateos, and A. Ribeiro (2017). "Network topology inference from spectral templates". *IEEE Transactions on Signal and Information Processing Over Networks*. 3(3): 467–483.

Sen, P., G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad (2008). "Collective classification in network data". *AI Magazine*. 29(3): 93–93.

Shi, J. and J. Malik (2000). "Normalized cuts and image segmentation". *Departmental Papers (CIS)*: 107.

Simon, H. A. (1991). "The architecture of complexity". In: *Facets of Systems Science*. Springer. 457–476.

Slawski, M. and M. Hein (2015). "Estimation of positive definite M-matrices and structure learning for attractive Gaussian Markov random fields". *Linear Algebra and Its Applications*. 473: 145–179.

Spielman, D. A. and S. H. Teng (2007). "Spectral partitioning works: Planar graphs and finite element meshes". *Linear Algebra and its Applications*. 421(2–3): 284–305.

Stanković, L. (2001). "A measure of some time–frequency distributions concentration". *Signal Processing*. 81(3): 621–631.

Stanković, L. (2015). *Digital Signal Processing with Selected Topics*. CreateSpace Independent Publishing Platform, An Amazon.com Company.

Stanković, L., M. Daković, D. Mandic, M. Brajović, B. Scalzo, and A. Constantinides (2020). "A low-dimensionality method for data-driven graph learning". In: *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 5340–5344.

Stanković, L., M. Daković, and E. Sejdić (2017). "Vertex-frequency analysis: A way to localize graph spectral components [Lecture Notes]". *IEEE Signal Processing Magazine*. 34(4): 176–182.

Stanković, L., D. P. Mandic, M. Daković, M. Brajović, B. Scalzo, and T. Constantinides (2019a). "Graph signal processing – Part I: Graphs, graph spectra, and spectral clustering". *arXiv:1907.03467*.

Stanković, L., D. P. Mandic, M. Dakovic, I. Kisil, E. Sejdic, and A. G. Constantinides (2019b). "Understanding the basis of graph signal processing via an intuitive example-driven approach [Lecture Notes]". *IEEE Signal Processing Magazine*. 36(6): 133–145.

Stanković, L., D. P. Mandic, M. Dakovic, and I. Kisil (2020). "Demystifying the coherence index in compressive sensing [Lecture Notes]". *IEEE Signal Processing Magazine*. 37(1): 152–162.

Stanković, L. and E. Sejdić (2019). *Vertex-Frequency Analysis of Graph Signals*. Springer.

Stanković, L., E. Sejdić, and M. Daković (2018). "Reduced interference vertex-frequency distributions". *IEEE Signal Processing Letters*. 25(9): 1393–1397.

Stanković, L., E. Sejdić, S. Stanković, M. Daković, and I. Orović (2019c). "A tutorial on sparse signal reconstruction and its applications in signal processing". *Circuits, Systems, and Signal Processing*. 38(3): 1206–1263.

Stoer, M. and F. Wagner (1997). "A simple min-cut algorithm". *Journal of the ACM (JACM)*. 44(4): 585–591.

Tanaka, Y. and Y. C. Eldar (2019). "Generalized sampling on graphs with subspace and smoothness priors". *arXiv preprint arXiv:1905.04441*.

Tanaka, Y. and A. Sakiyama (2019). "Oversampled transforms for graph signals". In: *Vertex-Frequency Analysis of Graph Signals*. Ed. by L. Stanković and E. Sejdić. Springer. 223–254.

Tang, J., M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei (2015). "Line: Large-scale information network embedding". In: *Proceedings of the 24th International Conference on World Wide Web*. 1067–1077.

Tang, L. and H. Liu (2011). "Leveraging social media networks for classification". *Data Mining and Knowledge Discovery*. 23(3): 447–478.

Tang, W., Z. Lu, and I. S. Dhillon (2009). "Clustering with multiple graphs". In: *Proceedings of the Ninth IEEE International Conference on Data Mining*. 1016–1021.

Thanou, D., D. I. Shuman, and P. Frossard (2014). "Learning parametric dictionaries for signals on graphs". *IEEE Transactions Signal Processessing*. 62(15): 3849–3862.

Thanou, D., X. Dong, D. Kressner, and P. Frossard (2017). "Learning heat diffusion graphs". *IEEE Transactions on Signal and Information Processing Over Networks*. 3(3): 484–499.

Transport for London (n.d.). URL: https://tfl.gov.uk/.

Ubaru, S., J. Chen, and Y. Saad (2017). "Fast estimation of tr(f(A)) via stochastic Lanczos quadrature". *SIAM Journal on Matrix Analysis and Applications*. 38(4): 1075–1099.

Veličković, P., G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio (2017). "Graph attention networks". *arXiv preprint arXiv:1710.10903*.

Verma, A. and K. K. Bharadwaj (2017a). "A comparative study based on tensor factorization and clustering techniques for community mining in heterogeneous social network". *In Proceedings of the International Conference on Computing, Communication and Networking Technologies (ICCCNT)*: 1–6.

Verma, A. and K. K. Bharadwaj (2017b). "Identifying community structure in a multi-relational network employing non-negative tensor factorization and GA k-means clustering". *Wires: Data Mining and Knowledge Discovery*. 7(1): 1–32.

Von Luxburg, U. (2007). "A tutorial on spectral clustering". *Statistics and Computing*. 17(4): 395–416.

Wagner, D. and F. Wagner (1993). "Between min cut and graph bisection". In: *Proc. of the International Symposium on Mathematical Foundations of Computer Science*. Springer. 744–750.

Wai, H.-T., Y. C. Eldar, A. E. Ozdaglar, and A. Scaglione (2019). "Community inference from graph signals with hidden nodes". In: *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 4948–4952.

Wang, H., J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo (2017). "GraphGAN: Graph representation learning with generative adversarial nets". *arXiv preprint arXiv:1711.08267*.

Wang, Z., A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli (2004). "Image quality assessment: From error visibility to structural similarity". *IEEE Transactions on Image Processing*. 13(4): 600–612.

Wu, Z., S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu (2019). "A comprehensive survey on graph neural networks". *arXiv preprint arXiv:1901.00596*.

Xu, Y. L. and D. P. Mandic (2020). "Recurrent graph tensor networks". *arXiv preprint arXiv:2009.08727*. Sept. arXiv: 2009.08727 [cs.LG].

Xu, Y. L., K. Konstantinidis, and D. P. Mandic (2020). "Multi-graph tensor networks". In: *Advances in Neural Information Processing Systems*.

Yankelevsky, Y. and M. Elad (2016). "Dual graph regularized dictionary learning". *IEEE Transactions on Signal and Information Processing Over Networks*. 2(4): 611–624.

You, J., R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec (2018). "GraphRNN: Generating realistic graphs with deep auto-regressive models". *arXiv preprint arXiv:1802.08773*.

Yu, W., C. Zheng, W. Cheng, C. C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang (2018). "Learning deep network representations with adversarially regularized autoencoders". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2663–2671.

Yuan, M. and Y. Lin (2006). "Model selection and estimation in regression with grouped variables". *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. 68(1): 49–67.

Yuan, M. and Y. Lin (2007). "Model selection and estimation in the Gaussian graphical model". *Biometrika*. 94(1): 19–35.

Zhang, J., X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung (2018a). "GAAN: Gated attention networks for learning on large and spatiotemporal graphs". *arXiv preprint arXiv:1803.07294*.

Zhang, J., X. Shi, S. Zhao, and I. King (2019a). "STAR-GCN: Stacked and reconstructed graph convolutional networks for recommender systems". *arXiv preprint arXiv:1905.13129*.

Zhang, M. and Y. Chen (2018). "Link prediction based on graph neural networks". In: *Advances in Neural Information Processing Systems*. 5165–5175.

Zhang, M., S. Jiang, Z. Cui, R. Garnett, and Y. Chen (2019b). "D-VAE: A variational autoencoder for directed acyclic graphs". In: *Advances in Neural Information Processing Systems*. 1588–1600.

Zhang, Z., P. Cui, and W. Zhu (2018b). "Deep learning on graphs: A survey". *arXiv preprint arXiv:1812.04202*.

Zhao, T., H. Liu, K. Roeder, J. Lafferty, and L. Wasserman (2012). "The huge package for high-dimensional undirected graph estimation in R". *Journal of Machine Learning Research*. 13(Apr): 1059–1062.

Zheng, M., J. Bu, C. Chen, C. Wang, L. Zhang, G. Qiu, and D. Cai (2011). "Graph regularized sparse coding for image representation". *IEEE Transactions on Image Processing*. 20(5): 1327–1336.

Zhou, D., J. Huang, and B. Schölkopf (2007). "Learning with hypergraphs: Clustering, classification, and embedding". In: *Advances in Neural Information Processing Systems*. 1601–1608.

Zhou, J., G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun (2018). "Graph neural networks: A review of methods and applications". *arXiv preprint arXiv:1812.08434*.

Zhu, X. J. (2005). "Semi-supervised learning literature survey". *Tech. rep.* University of Wisconsin-Madison, Department of Computer Sciences.