1 Introduction

Graph signal processing (GSP), or **signal processing on graphs**, is the study of how to analyze and process data associated with graphs. Graphs can represent networks, including both physical networks (the Internet, sensor networks, electrical grids or the brain) and information networks (the world wide web, Wikipedia or an online social network). Graphs can also be used to represent data (pixels in an image, or a training dataset used for machine learning). In this chapter we start by defining graphs and graph signals (Section 1.1). We develop some intuition about how to quantify signal variation on a graph, which will later be used to introduce the concept of graph signal frequency. We give examples of simple operations that can be performed on graphs, such as filtering or sampling. An important feature of GSP is that processing tools can be adapted to graph characteristics (Section 1.2). We present examples of graphs encountered in conventional signal processing (Section 1.3) and this is followed by a discussion of graphs arising in some representative applications (Section 1.4) and a brief overview of mathematical models for graphs (Section 1.5). We conclude the chapter with a roadmap to the rest of the book (Section 1.6).

1.1 From Signals to Graph Signals

1.1.1 Graphs

A **graph** consists of a series of **nodes**, or vertices, whose relations are captured by **edges** (see Figure 1.1). Two nodes are said to be **connected** when there exists an edge between them. Figure 1.1 shows a fully connected three-node graph, where all nodes connect with each other. Depending on the application, nodes and edges may correspond to components in a physical network (e.g., two computers connected by a communication link), entries in an information network (e.g., two linked Wikipedia pages) or different items in a dataset (e.g., two similar items used for training a machine learning system).

More formally, we define a graph as a set of nodes or vertices, $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ or $\mathcal{V} = \{1, 2, \ldots, N\}$[1], and a set of edges, $\mathcal{E} = \{e_1, e_2, \ldots, e_M\}$ or $\mathcal{E} = \{1, 2, \ldots, M\}$. We may also denote an edge as $e = v_i v_j$ if it connects nodes i and j. The edges can be **weighted**, so that a scalar value is associated with each edge, or **unweighted**, i.e., all edges have weight equal to 1. With each node we associate a **degree**, the total weight

[1] We will use the terms vertex and node interchangeably throughout this book.

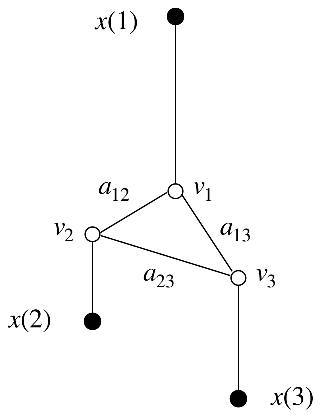


Figure 1.1 Simple example graph with three nodes, v_1, v_2, v_3, and edge weights a_{12}, a_{13}, a_{23}. There is a graph signal associated with this graph, with values $x(1), x(2), x(3)$ respectively at nodes v_1, v_2 and v_3. We will consider weights that are positive and capture the proximity or similarity between the nodes. For example, if $a_{12} > a_{13}$ then the first node, v_1, is closer (or more similar) to the second node, v_2, than to the third node, v_3.

of the edges connecting to that node. Some graphs are **directed**: only one of $v_i v_j$ and $v_j v_i$ may exist, or they may both exist but have different edge weights. Other graphs are **undirected**: $v_i v_j$ and $v_j v_i$ both exist and both edges have the same weight if the graph is weighted. The choice between a directed or an undirected graph will depend on the application, as shown in the following examples.

Example 1.1 Directed graph If A and B represent two locations on a map, and we choose an edge weight based on distance, it makes sense for the edge to be undirected, since the distance between A and B is the same as the distance between B and A. On the other hand, if we are considering the distance between those two points following an existing road network, the two distances may be different, as illustrated in Figure 1.2.

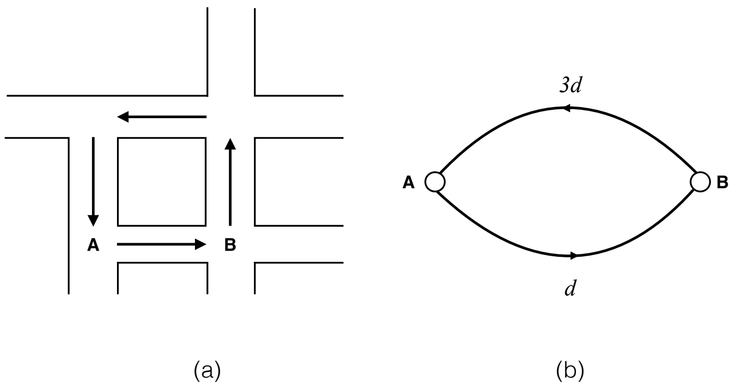


Figure 1.2 Directed road network graph. (a) City area with one-way streets, where driving distances on the road network are not symmetric. (b) Corresponding directed graph with different weights in each direction.

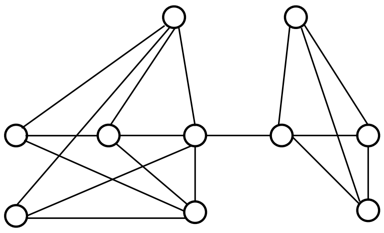


Figure 1.3 Undirected social network graph: Users have to agree to connect to each other and thus connections are undirected. It is common for connected users to share multiple common connections.

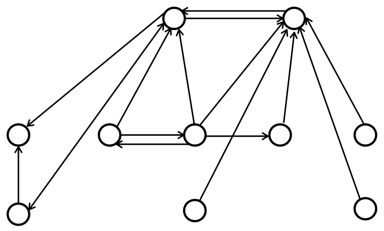


Figure 1.4 Example of a directed social graph. Some users attract many followers while others do not have any. Sometimes there are connections in both directions.

Example 1.2 Undirected graph In some social networks two users have to agree to be connected, while in others one user might follow another, but not the other way around. The first type of social network leads to an undirected graph (Figure 1.3), while the second should be represented by a directed graph (Figure 1.4). The questions of interest and properties are in general quite different for these two types of social graphs. In an undirected social graph connected users often share common connections which leads to a "clustering" behavior: we can observe groups of users with strong connections. In the directed social network case, a typical characteristic is for some users to have many connections, while most users have very few.

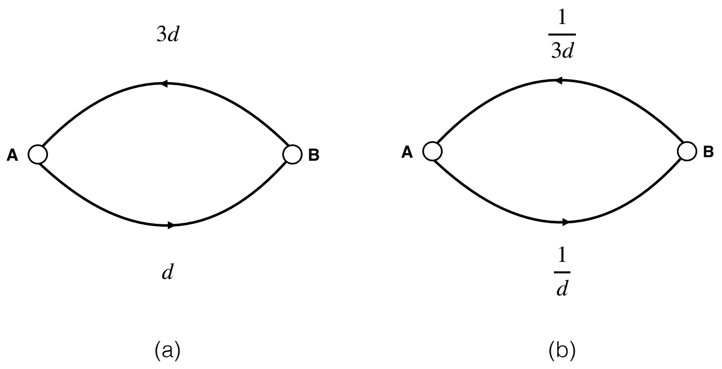


Figure 1.5 (a) Directed distance version of Figure 1.2, where larger weights indicate longer distance. (b) In a similarity graph, where, as in this example, weights are inversely proportional to distance, nodes that are closer are more likely to be similar.

Example 1.3 Distance graphs and similarity graphs In Figure 1.2, edge weights represent distance, i.e., a larger weight corresponds to nodes that are farther apart. In contrast, in a similarity graph, larger weights correspond to greater similarity and smaller distance, as shown in Figure 1.5. Throughout most of this book we will use similarity graphs, with the assumption that signals observed at two connected nodes will tend to have similar values if the edge weight between those nodes is large.

Most weighted graphs studied in this book are *similarity* graphs, where the (positive) edge weights quantify how similar two nodes are (larger weights mean greater similarity). For example, a larger weight may be chosen between two nodes in a sensor network if those two nodes are close to each other, and therefore their measurements are more likely to be similar. This is illustrated in Example 1.3, where a graph has weights representing distance, while a second graph with the same topology represents similarity.

More generally, the meaning given to the existence of an edge between two nodes, i and j, depends on the application. A edge could be used to indicate that "i is at certain distance from j" or that "i may happen if j happened" or simply that "A and B are similar". Thus, graph representations are general. They provide a language and associated mathematical tools for very different problems and applications.

1.1.2 Frequency Analysis of Graph Signals

Graph signals This book is about processing signals defined on graphs. Assume that each node i in a graph has a scalar value $x(i)$ associated with it.[2] The aggregation of all these scalar values into a vector $\mathbf{x}$ of dimension $|\mathcal{V}| = N$ will be called a **graph signal**. As an example, the graph signal in Figure 1.1 takes values $x(1), x(2), x(3)$ corresponding to nodes v_1, v_2, v_3, respectively. For a given graph, there can be many different graph signals. As an example, a set of sensors can make different daily measurements, each leading to a different graph signal.

What we define to be a graph signal depends on the application: in a graph representing physical sensors, graph signals could be measurements (e.g., temperature); in a graph representing a machine learning dataset, the signal might represent the label information; a graph associated with an image may have pixel intensities as graph signals; and so on. Concrete examples of graph signals are discussed in Section 1.4.

Frequency and variation Frequency is a key concept in conventional signal processing. Informally, terms such as "low frequency" or "high frequency" are commonly used in everyday conversation when referring to music and audio signals. A more mathematical definition of frequency, with complex sinusoids representing elementary frequencies, would be familiar to many science and engineering students. The definition of frequency, the number of cycles per second, indicates that, loosely speaking, higher frequency signals change faster, i.e., have greater variation, than signals with lower

[2] Generalization to the case where vectors are associated with each node is possible, but is not explicitly considered in this book.

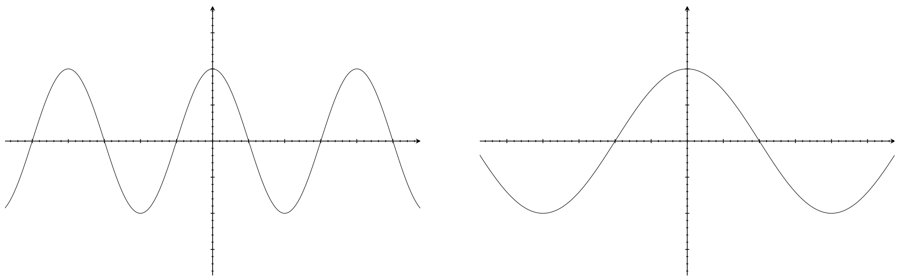


Figure 1.6 The sinusoid on the left has higher frequency than than on the right (the respective periods are 2 and 4). This can also be viewed in terms of variation. For example, selecting a small interval length Δt, say $\Delta t = 0.1$, we can observe that with any interval $[t, t + \Delta t]$ there is in general more change, i.e., more variation, for the sinusoid on the left. For graphs with arbitrary connections between edges and nodes, the focus of our interest, we will not be able to easily plot a graph signal or define frequencies in terms of sinusoids, but we will use the notion of variation to define elementary frequencies.

frequencies. This is illustrated in Figure 1.6. Since it is not possible, except for very particular cases, to have a meaningful definition of sinusoids for a graph with arbitrary nodes and edges, we define the concept of frequency for graph signals by quantifying the **graph signal variation**: greater variation on the graph leads to higher frequency.

Defining variation for time signals Before considering graph signals, let us take a closer look at a conventional discrete-time signal, where each sample is associated with a point in time (e.g., speech, audio). We can reason in a similar way with images, where each pixel represents color or intensity in space. Taking the time-based signal of Figure 1.7(a) as an example, we can quantify how fast the signal changes, i.e., how large its **variation** is. Let us do this by computing the difference between two consecutive samples as $|x(k+1)-x(k)|$, where the absolute value is used since increases and decreases are equivalent in terms of variation. Notice that if we multiply all the samples $x(k)$ by some constant C, the overall variation would be multiplied by C. Thus, if we wish to compare two different signals in terms of variation we need to normalize the signals or, equivalently, normalize the variation. Then define a normalized **total variation**:

$$TV_t = \frac{1}{\sum_k |x(k)|} \sum_k |x(k+1) - x(k)|. \tag{1.1}$$

Notice that in Figure 1.7(a) the samples are equispaced, as is the case for conventional signals. Thus, in an audio signal with N samples per second, the interval between two consecutive samples is $1/N$ seconds. Similarly, the distance between neighboring pixels in an image is always the same.

Variation for irregularly spaced time signals In order to develop a metric of variation for graph signals, let us consider now the case where we have samples at arbitrary

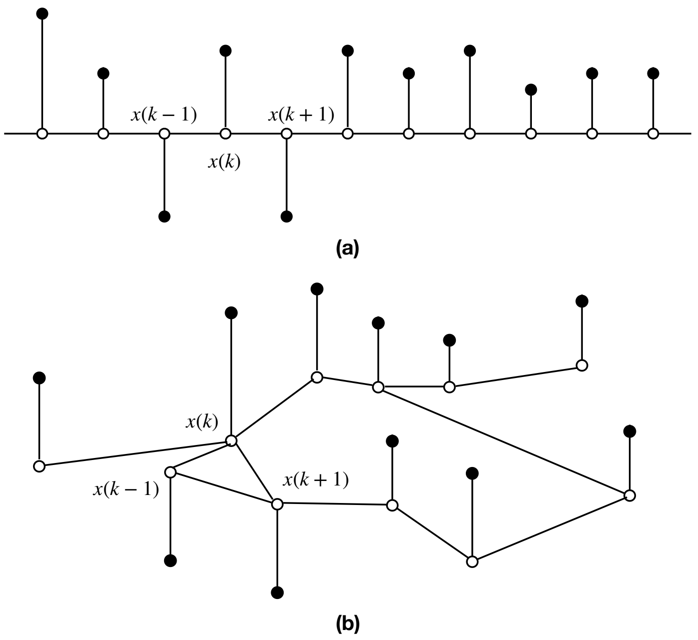


Figure 1.7 Variation and frequency in the time and node domains. The same signal seen as (a) observations in time and (b) observations placed on a graph. The signal in (a) can be viewed as being positioned on a line graph, while in (b) we have an arbitrary graph.

times $k_1, k_2, \ldots$ (e.g., due to missing data in the observations). Then, our definition can be adapted to measure "variation per unit time," that is:

$$\frac{1}{(k_{i+1} - k_i)} |x(k_{i+1}) - x(k_i)|$$

where the variation between samples further apart in time counts less. Then we can define the **normalized total variation** as

$$TV_t = \frac{1}{\sum_i |x(k_i)|} \sum_i \frac{1}{(k_{i+1} - k_i)} |x(k_{i+1}) - x(k_i)|, \tag{1.2}$$

which gives us (1.1) when $k_{i+1} - k_i = 1$ for all i.

Defining variation for graph signals To define variation for graph signals, let us follow the same principles we used to define (1.2): (i) compute the absolute difference between the signal value at *neighboring nodes* and (ii) scale this measure of variation by the *distance between nodes*.

Referring to the graph example in Figure 1.7(b), note that the node index has no meaning, i.e., the difference between the indices of two nodes v_1 and v_2 (i.e., $1 - 2$) cannot be interpreted as a time, as in the example in Figure 1.7(a). The example in Figure 1.5(b) showed that one can define a notion of the **similarity** between nodes as the inverse of their distance. Thus, rather than divide variations by distance, as in (1.2), let us *multiply them by the similarity*. Then the definitions of variation for graphs will be based on elementary operators such as

$$a_{12}|x(v_1) - x(v_2)| = \frac{1}{d_{12}}|x(v_1) - x(v_2)|$$

where the difference between the signals at connected nodes v_1 and v_2 is weighted by their similarity, a_{12}. Note that when distance becomes arbitrarily small its reciprocal becomes arbitrarily large. In practice, however, similarity is often derived from distance using a kernel function, such as an exponential, so that the maximum similarity is 1 as the distance between nodes becomes small.

With this definition the variation can be quantified for any graph by considering only nodes that are neighbors (i.e., such that their similarity is non-zero). Thus, the metric of (1.2) can be extended in a natural way as follows:

$$TV_g = \frac{1}{\sum_i |x(v_i)|} \sum_{i \sim j} a_{ij}|x(v_i) - x(v_j)|, \tag{1.3}$$

where the sum is over all pairs of nodes i and j that are connected, denoted by $i \sim j$, corresponding to $a_{ij} \neq 0$. The intuition behind this definition is illustrated in the following example. The problem of defining frequencies for graph signals will be studied in detail in Chapter 3.

Example 1.4 Time variation and graph variation Compare Figure 1.7(b) with Figure 1.7(a). The nodes v_k, v_{k-1}, v_{k+1} have the same values in Figure 1.7(a) and in Figure 1.7(b). But in Figure 1.7(b) the connections between these nodes (and between those and other nodes) are somewhat more complicated. For example, unlike in Figure 1.7(a), nodes v_{k-1} and v_{k+1} are connected. Also, node v_k is connected to two other nodes apart from v_{k-1} and v_{k+1}, and since these other nodes have values closer to $x(k)$ the frequency of the signal on the graph of Figure 1.7(b) may be lower since the variation is smaller.

Even a simple definition of graph variation such as that of (1.3) can be useful in practice. For example, if we assume that data variation should be small under normal circumstances, the definition (1.3) can be used to detect anomalies corresponding to excessive variation. This is illustrated by the following example.

Example 1.5 Sensor networks – anomaly detection Consider a set of temperature sensors inside a large factory. The goal is to use temperature measurements to determine whether there may be problems with some of the equipment. As an example, equipment overheating could be observed through locally higher temperature measurements.

Construct a graph where each node represents a sensor (and measurement). In order to detect an "anomaly" we can compare temperatures in different parts of the factory floor and try to identify unexpected behavior. Each graph node is associated with a sensor, so we need to decide how the nodes should be connected, i.e., how to choose edge weights between pairs of nodes.

We may consider the situation to be anomalous if a sensor measurement is very different from that of its close neighbors. Intuitively, it may be normal to measure very different temperature in areas far away from each other, but if two sensors are nearby and the temperature is much higher in one sensor, this could be an indication of equipment (or sensor) malfunction. Thus, we can consider a graph with edges having weights that are decreasing functions of the distance, and where only nodes close enough to each other will be considered to be connected. A simple detection strategy may then involve comparing the signal values at a node with those in the immediate neighborhood.

1.1.3 Filtering and Sampling Graph Signals

Conventional signals can be represented as a linear combination of elementary signals representing frequencies (e.g., complex sinusoids in the case of the Fourier transform). These frequencies can be ordered, so that we can talk about high and low frequencies. Similarly, we can develop graph signal processing for a given graph by defining frequency in a way that takes into account the characteristics of that graph.

This idea of quantifying signal variation across nodes will lead us to introduce frequency representations for graph signals, with high frequencies corresponding to fast signal variation across connected nodes. This notion of frequency will allow us to define tools to analyze, filter, transform and sample graph signals in a way that takes into account the signal variation on the graph. In terms that would be familiar to a signal processing practitioner, these tools will allow us to talk about “low pass” graph signals, with corresponding “low pass” filters. To address the sampling problem, we would like to be able to define the required level of “smoothness” that a signal is required to have in order for it to be recovered from a set of signal samples (i.e., observations made at a subset of nodes).

Example 1.6 Sensor networks – denoising Considering again Example 1.5, we can formulate the problem of denoising the observed data using the concept of the graph signal frequency. Removing noise that affects measurements requires making assumptions about the signal measured and the measurement noise. For example, if we expect the temperature measurements to be similar across neighboring nodes we may use this to design a low pass filter that can reduce noise in the temperature measurements before processing them. A simple example of this could be a filter that updates the value at each node using a weighted average of the values at neighboring nodes.

Example 1.7 Sensor network – power management Assume that it is necessary to turn off some sensors to reduce energy consumption. We can formulate the problem of selecting which sensors to turn off on the basis of the graph signal frequency, by viewing this as a sampling problem, Then, the goal is to select a subset of sensors under the assumption that the values at sensors that were not observed can be estimated using interpolating filters. Graph signal sampling will take into account the relative position of the nodes (the graph structure) to select the most informative nodes (sensors).

1.1.4 Graph Signal Processing versus Vector Processing

For a graph with N nodes we can combine all the $x(v)$ into a single vector $\mathbf{x} \in \mathbb{R}^N$. Thus, it is worth asking why we could not simply work with $\mathbf{x}$ as a vector in $\mathbb{R}^N$ and apply existing methods from linear algebra to transform and analyze this vector.

To answer this question, first note that for regular domain signals the sample indices are meaningful, e.g., in the time domain we know that sample $x(k+1)$ comes after sample $x(k)$. In contrast, for graph signals the indices associated with each node are arbitrary. Thus, for two nodes i and j, the indices i and j themselves do not matter; what matters is whether there is an edge between those nodes. We can change the labels but, as long as the connections in the graph remain the same, the tools we develop for graph signal analysis will be the same (a simple permutation of the input signal). In general, for a graph with arbitrary connectivity, there is no obvious way to use a standard transform such as the DFT, given that there are many different ways of mapping a graph signal into a vector along a line. For example, given the signal of Figure 1.7(b) there are many ways of mapping it into a vector along a line as in Figure 1.7(a). Second, the same signal may have very different interpretations depending on how the nodes are connected. This is really the main motivation for graph signal processing and can be illustrated with a simple example.

Example 1.8 Consider the two graphs $\mathcal{G}$ and $\mathcal{G}'$ in Figure 1.7(a) and (b). The same graph signal is associated with both graphs. For which of those two graphs does the signal have higher variation?

Solution

Comparing Figure 1.7(a) and Figure 1.7(b), we observe that there are only two nodes with negative values. Note that those two nodes have only one connection to others with positive values in Figure 1.7(a), while several such connections exist in Figure 1.7(b). From this, we may infer that the signal has a higher frequency for the graph of Figure 1.7(b).

From Example 1.8 we see that the same graph signal can be associated with two different graphs (with the same nodes but different edges). Processing this signal as a graph signal is different from treating it as a vector because the tools we use for processing (e.g., our definition of frequency) depend on the graph. This idea is further developed next.

1.2 GSP Tools Adapt to Graph Characteristics

An important feature of graph signal processing is that it provides a *common* framework to study systems that are fundamentally different in their behavior and properties.

1.2.1 Selecting the Right Graph for a Task

In data science applications the goal is to extract useful information from data. To that end, GSP methods process and analyze data taking into account the topology of an underlying graph. However, a given signal $\mathbf{x}$ can be interpreted in different ways depending on the graph with which it is associated, as seen in Example 1.8.

In some cases the choice of graph is obvious given the application, while in others choosing the "right" graph is key to achieving meaningful results. We will discuss both types of scenarios, including techniques to optimize the graph selection, in Chapter 6. The problem of selecting a graph for a specific problem is illustrated in Example 1.9.

Example 1.9 In Example 1.5 assume that the temperature sensors are deployed inside a building, for which we have a floor plan. A straightforward definition of a graph would be simply based on the physical distances between the sensors, as discussed in Example 1.5. Given that variation (and thus frequency) depends on how the graph is connected, suggest alternative ways in which a graph could be built in order to provide a better analysis of the temperature within the building.

Solution

In Example 1.5 we were trying to detect anomalous behavior by identifying sensors whose temperature measurements are very different from those of neighboring sensors, since we expect that temperatures will generally be uniform within a room but may change from room to room. A possible solution is to select edge weights that decrease with distance but also take into account the presence of walls in between the corresponding sensors.

1.2.2 Graph Diversity: How General Are GSP Tools?

Whether a graph is given or has to be selected, we can encounter significant diversity in the graph properties. Graph sizes can vary significantly, from hundreds of nodes (e.g., in a sensor network) to millions of nodes (e.g., in an online social network). Similarly, the graph topology can be very regular (all nodes have the same number of connections) or highly irregular (some nodes have orders of magnitude more connections than others). Graphs representing online social networks can be highly irregular (e.g., some users have millions of followers, others few), while other graphs can be designed to be very regular (e.g., a K-nearest-neighbor graph connects each point in space to exactly the K closest points in space). Graphs can be directed or undirected (see Example 1.3), weighted or unweighted. Thus, even though we will be defining a single set of mathematical tools to work with many different types of graphs we should always keep in mind the following remark.

Remark 1.1 In this book we develop tools that are meant to be generic and applicable to graphs with different characteristics. However, interpretation and insights derived from processing may be quite different from case to case. The tools may be generic, but their interpretation is highly dependent on graph properties.

Throughout this book, our goal will be to strike a balance between generality and specific behavior, by describing the general ideas first and then explaining how they apply to specific graphs. In what follows we present several examples of graphs and graph signals to illustrate the diversity of cases we may encounter. These examples will be used again in later chapters to provide more insights about the concepts we introduce.

For each of these examples we explain what the nodes and edges are and provide examples of graph signals of interest. We also include in our discussion the **degree distribution**, which measures how different the nodes in the graph are from each other. This is based on computing the degree of each node (its total number of edges if the graph is unweighted, or the sum of all its edge weights if the graph is weighted) and then characterizing how the degree changes across the graph. For example, a regular graph would be such that all its nodes have (approximately) the same degree. The examples in the following sections were generated with `Matlab` using the `GraSP` toolbox, which will be used throughout the book and is introduced in more detail in Appendix B. The code used to generate many figures in the book is available from the book's web page.[3].

1.3 Graphs in Classical Signal Processing

We start by introducing graphs that are closely linked to conventional signal processing, namely, path,[4] cycle and grid graphs, which correspond to finite 1D and 2D discrete signals. Frequency representations for these graphs match exactly those available for the corresponding signals (1D and 2D) studied in classical signal processing. This allows us to view GSP tools as a "natural" generalization of existing methods.

1.3.1 Path Graphs, Cycle Graphs and Discrete-Time Signals

Discrete-time (and finite-length) signals can be interpreted as graph signals. We do this by associating one graph node to each time instant, and letting a sample in time be the signal associated with the corresponding node. For a finite-length signal we can choose the path graph of Figure 1.8, generated using Code Example 1.1. If the similarity between neighboring nodes in Figure 1.8 is 1, then the total variation of (1.2) and the

[3] `http://www.graph-signal-processing-book.org/`
[4] We use the term path graph, rather than line graph. In graph theory, the term line graph is reserved for a graph derived from an existing graph, where edges in the original graph become nodes in the line graph.

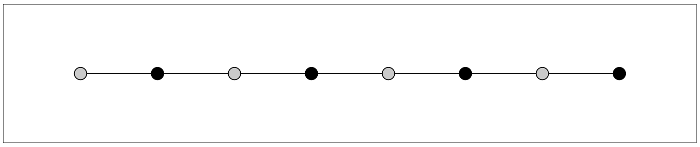


Figure 1.8 Define a path graph with N nodes and with all edge weights equal to 1. Then, a length-N time signal, where consecutive samples in time are associated with consecutive nodes on the graph, can be viewed as a graph signal. In this case the *graph frequency* definitions will correspond to the conventional definitions for time signals.

Code Example 1.1 `Matlab` code used to generate Figure 1.8

```
path10 = grasp_non_directed_path(8);
signal10 = [200 10 200 10 200 10 200 10]';
grasp_show_graph(gca, path8,...
                 'node_values', signal8,...
                 'color_map', 'gray',...
                 'value_scale', [0 255]);
ylim([4 6]);

save_figure('path8');
```

graph total variation of (1.3) are equal. The graph frequency definitions that we will develop in Chapter 3 can be shown to correspond exactly to the discrete cosine transform (DCT)[1].

The same finite-length signal can also be mapped to a cycle graph (Figure 1.9). Since two nodes connected in the graph are consecutive in time, a cycle can be viewed as a representation of an infinite-length periodic signal with period N, the number of nodes. The graph of Figure 1.8 is almost regular, i.e., it has the same connectivity for each node except the two end nodes, while the cycle graph of Figure 1.9 is exactly regular. In this case, as will be seen in Chapter 3, the frequency representation associated with the graph is the discrete Fourier transform (DFT).

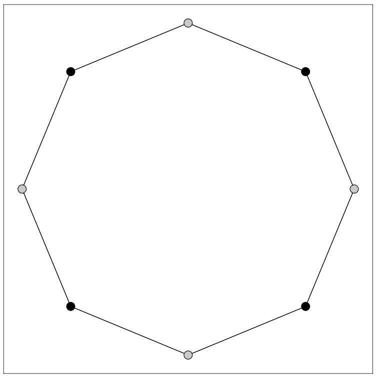


Figure 1.9 Cycle graph with same graph signal as in Figure 1.8. Note that here the two end-points of Figure 1.8 are connected. This is equivalent to viewing this signal as containing eight samples of a periodic signal with period eight.

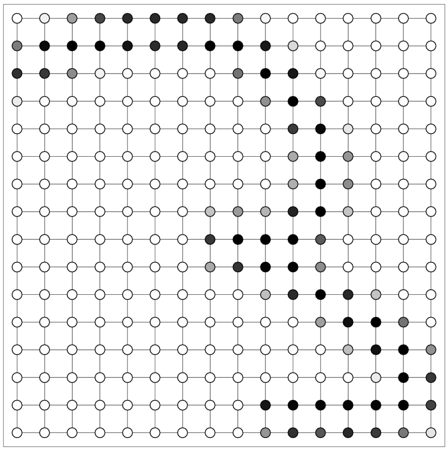


Figure 1.10 Grid graph representing an image from the USPS dataset. Each node corresponds to a pixel, and the signal at each node is the intensity of the corresponding pixel. In a signal-independent graph all edge weights can be equal.

The same signal is associated with two different graphs in Figure 1.8 and Figure 1.9, leading to different interpretations. Specifically, on the path graph of Figure 1.8, nodes 1 and N are not connected, so that the difference between $x(v_1)$ and $x(v_N)$ does not affect the total variation (1.3). In contrast, on the cycle graph of Figure 1.9, nodes 1 and N are connected (corresponding to the connection of consecutive cycles in a periodic signal) and thus $|x(v_1) - x(v_N)|$ will contribute to the total variation in (1.3). Comparing these two graph choices also reminds us that conventional signal processing relies on having multiple different representations for the same signal (e.g., DCT and DFT, among many others), with the choice of representation dependent on the specific application.

In Box 1.1 we summarize the basic ideas of 1D signals and their associated graphs. Throughout this chapter we will use similar boxes to summarize the properties of other graphs of interest. Each box includes a description of what the nodes are, and how their edges and edge weights may be chosen. We also mention whether the graph is regular (each node has roughly the same number of neighbors), describe the signals of interest and the typical scale of those graphs and give example applications.

Box 1.1 1D signals

- **Nodes**: one per signal sample
- **Edges**: only between samples that are neighbors in time
- **Degree distribution**: regular (each node has two neighbors) or nearly regular (in the line graph case, the two end nodes only have one neighbor each)
- **Signal**: the values of each sample in time
- **Scale**: each finite set of samples can be viewed as a "window" for time signal analysis; typical windows could be a few hundred to a few thousand samples
- **Applications**: audio, speech processing, for example

1.3.2 Images and Grid Graphs

Two-dimensional images can be interpreted as graph signals by defining a regular grid graph, where each node, corresponding to a pixel, can be connected to its four immediate neighbors. Pixels at the image boundaries can have two or three neighbors, as shown in Figure 1.10. The frequency modes for this grid graph are obtained from the 2D separable DCT.

As an alternative, we can also consider a grid graph that is exactly regular (all nodes have the same number of neighbors) by connecting row and column path graphs as cycles. Then each row (or column) of the grid graph corresponds to a path graph (similar to Figure 1.8) and becomes connected as a cycle (similar to Figure 1.9). Thus we connect the left and right nodes of each horizontal path graph, and similarly the top and bottom nodes of each vertical path graph. Mathematically, assume that pixels are indexed by $x(i, j)$ where $i, j = 0, \dots, N-1$. Then in a 4-connected graph, a given pixel would be connected to four neighbors: $x((i+1) \bmod N, j)$, $x(i, (j+1) \bmod N)$, $x((i-1) \bmod N, j)$, and $x(i, (j-1) \bmod N)$, where the modulo operation allows nodes along the top row and right column to be connected to the corresponding nodes along the bottom row and left column, respectively. For example, the pixel at the top right corner will have as a neighbor to its "right" the pixel at the top left corner, and its neighbor "above" will be the bottom right pixel. The frequency representation for this graph is the 2D separable DFT.

Box 1.2 2D images

- **Nodes**: one per pixel
- **Edges**: only between pixels that are neighbors in space
- **Degree distribution**: regular
- **Signal**: intensity (or color information) at each pixel
- **Scale**: total number of nodes is equal to the number of pixels (typically in the order of millions)
- **Applications**: image processing

1.3.3 Path and Grid Graphs with Unequal Weights

While signals associated with the graphs of Box 1.1 and Box 1.2 can already be analyzed using existing signal processing methods, these examples are still important. These representations lead to graph-based tools corresponding to well-known transforms, such as the 8×8 DCT for the path graph of Figure 1.8, which has been widely used, for example in the JPEG image compression standard [2]. But slight changes to those weights lead to different transformations (see Section 7.4). As graphs deviate from the properties used in conventional signal processing (e.g., when the edge weights are no longer equal), we can observe how the corresponding signal representations change; this provides insights about how the tools we develop adapt to changes in graph topology. For example, if all edge weights on the path graph are equal to 1 except for $a_{i,i+1}$ then this serves to indicate that samples up to i are somewhat decoupled from those after i. In the context of

Table 1.1 Classes of problems of interest

Category	Nodes	Links	Example
Physical networks	Devices	Communication	Sensor networks
Information networks	Items	Links	Web
Machine learning	Data examples	Similarity	Semi-supervised learning
Complex systems	System state	System transitions	Reinforcement learning
Social networks	People	Relationships	Online social networks

image processing this can be used to signal the presence of an image contour between i and $i + 1$. This simple observation can be seen to be a basic principle behind bilateral filtering, a highly popular image-dependent image filter [3].

1.4 Graphs Everywhere?

As technology for sensing, computing and communicating continues to improve, we are becoming increasingly reliant on a series of very large scale networks: the Internet, which connects computers and phones as well as a rapidly growing number of devices and systems (the Internet of Things); large information networks such as the web or online social networks. Even networks that have existed for decades (e.g., transportation or electrical networks) are now more complex and increasingly a focus of data-driven optimization. In what follows we describe examples of systems that can be understood and analyzed by using a graph representation, and to which GSP tools can potentially be applied. Our aim is not to be exhaustive, but rather to illustrate the wide variety of applications that can be considered. We divide these examples into several categories, which we discuss next (see Table 1.1).

1.4.1 Physical Networks

A physical network is a system where there are devices or components that can be represented as nodes in the graph model. The edges between two nodes are a function of distance or represent a physical communication link. For example, in a transportation network, the nodes may represent hubs or intersections while the links (edges) may correspond to roads or train tracks. In an electric network the nodes may include power generators and homes, while the links would represent transmission lines. In a communication network such as the Internet, each node may be a computer or other connected device, while each physical link between devices would represent a communication link.

The transportation road network in Figure 1.11 demonstrates one key feature of many of these physical networks: node position corresponds to an actual location in space. In some cases, e.g., sensor networks such as that of Figure 1.12, there are no obvious links between the nodes (the sensors), and thus one may construct a graph as a function of distance. In others, e.g., road networks, a better choice might be to select edge weights

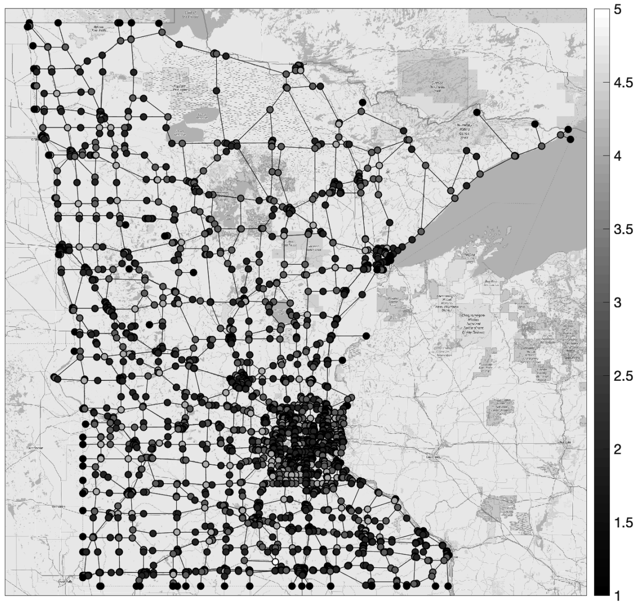


Figure 1.11 Minnesota road network graph. The signal value associated with each node on the graph is the corresponding node degree. Note that the node degrees in the graph are low, reflecting the physical limitations of the road network.

that are a function of the distance between nodes when following the network (i.e., the distance along the road, rather than the distance between nodes). This was illustrated in Figure 1.2. Finally, in communications networks, e.g., the Internet, the exact position of the nodes may be known but may not be relevant to define a graph representation. Instead, the capacity of the communication links may be more important. When analyzing a physical network, the properties of the graph topology may be constrained by those of the system itself. For example, in a realistic road network the number of roads leading to an intersection cannot be arbitrarily large, and thus the corresponding graphs will have low degree (see Figure 1.11). Also, as shown in Figure 1.2, one-way streets in a transportation network lead to directed graphs.

Box 1.3 Sensor network

- **Nodes**: one per sensor
- **Edges**: only between sensors that are within a certain range of each other (e.g., radio range)
- **Degree distribution**: can be regular, e.g., *k*-nearest-neighbor graph
- **Signal**: a set of sensor measurements, for example of temperature
- **Scale**: dozens to thousands of nodes
- **Applications**: environmental monitoring, anomaly detection

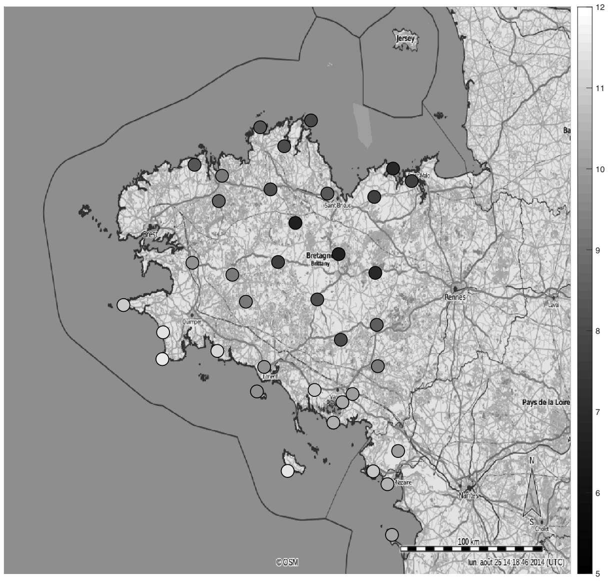


Figure 1.12 Sensor network example: weather stations. Note that here we know only the positions of the sensors. We could create a graph by connecting each sensor to its nearest neighbors. A graph signal is a set of temperatures (Celsius) measured at each weather station at a given time.

Sensor networks A sensor network is a concrete example of a physical system (see Box 1.3). A variety of systems can be described as sensor networks, including sets of distributed temperature sensors (indoors or outdoors), surveillance cameras or even devices carried by users, such as cellphones. Sensor networks are often deployed to sample information from the real world, e.g., temperature. Thus, the nodes are likely to have a location in space, and edge weights between nodes can be computed as a function of distance. The example of Figure 1.12 shows a regional network of weather sensors. Sensors can be deployed at different scales, within a building or on a bridge, across a city, or a region, and can gather many different types of measurements (temperature, air quality, vibrations, etc.) The processing of data obtained from sensor networks is an important motivating application for GSP. Data acquired by a sensor network is very often irregularly spaced. The methods we consider here will allow us to take into account sensor location as part of the processing, as illustrated in Examples 1.5, 1.6 and 1.7.

1.4.2 Information Networks

Organization of information as a network or graph is not new (e.g., encyclopedias are organized with entries and cross-references). The Internet has made this way of organizing information more explicit (and easier to navigate). Indeed, one of the most popular sources of information is Wikipedia, which is in itself an online version of an encyclopedia. In typical information networks each node represents an item of information (a web page, a Wikipedia entry) while (directed) links correspond to linking and cross-referencing between the items.

The web can be easily seen as a graph (see Box 1.4) where each page corresponds to a node, and the graph is directed, since one page can link to another without the linking being reciprocal. This graph structure has often been used to analyze page content. As an example, we can consider blogs dealing with specific topics (e.g., politics) and establish how they link to each other. Then a graph signal may be created at each node (blog) by creating a histogram of the frequency of appearances of certain keywords in that blog.

Box 1.4 The web

- **Nodes**: one per web page
- **Edges**: between web pages that reference each other
- **Degree distribution**: highly irregular
- **Signal**: some quantitative information extracted from a particular web page or blog (e.g., a distribution of the frequency of occurrence of certain keywords)
- **Scale**: the number of web pages is estimated to be several billion
- **Application**: information search

Note that graph-based representations of the web were key to the original PageRank search algorithm, where the essential idea was that the most relevant pages in a search would be those more likely to be traversed through a random walk of all pages that contain the search term [4].

1.4.3 Machine Learning

Supervised classification is a machine learning problem where the goal is to assign labels to data. For example, we may have a collection of images belonging to some categories (dogs or cats, say) and we would like to automate the process of determining to which category a new image belongs. An initial step in this design is to collect a representative set of labeled images, i.e., a training set containing examples of all categories under consideration. Then it is useful to consider the **similarity graph** associated with this training set. To do this, each image is mapped to a feature vector, which could be formed by the pixel values, or some information derived from the pixel values. In this graph each node corresponds to a data point, i.e., one image in the training set, and the edge weights are a function of the similarity between two data points, i.e., how similar the two images are in the chosen feature space. A typical approach to define a similarity graph is to associate a function of the form

$$a_{ij} = \exp(-d(\mathbf{x}_i, \mathbf{x}_j)^2/\sigma^2)$$

with the distance between two data points $\mathbf{x}_i$ and $\mathbf{x}_j$. Thus, maximum similarity ($a_{ij} \approx 1$) is achieved when the two objects (for example, images) are close to each other. If the choice of distance metric is good, we expect the similarity between objects in different classes to be low (and $a_{ij} \approx 0$ if $d(\mathbf{x}_i, \mathbf{x}_j) \gg \sigma^2$).

Box 1.5 Learning: similarity graph

- **Nodes**: one per data point used in the learning process
- **Edges**: between data points as a function of their distance in feature space
- **Degree distribution**: can be regular, e.g., k-nearest-neighbor graphs
- **Signal**: label for each data point
- **Scale**: the number of nodes corresponds to the size of the training set, which could be in the order of millions of points for some modern datasets
- **Application**: classifier design, semi-supervised learning

Graph-based methods have been used for unsupervised clustering, where the goal is to find a natural way to group data points having the same label. Notice that if a good feature space has been chosen (and the classification problem is relatively simple) one would expect neighboring nodes on the graph to have the same label. We will view this as a "smoothness" associated with the label signal. We will introduce this notion more formally and apply it to learning in Chapter 6.

1.4.4 Analyzing Complex Systems

In a complex system, many actions and events can affect overall behavior and performance. Examples of these systems include communication or transportation networks, a manufacturing plant, and a set of autonomous agents. A common problem in such systems is to take actions, often in a distributed way, in order to achieve some overarching performance goals. Note that, even in cases where there is an underlying physical system, e.g., a transportation network, we are simply interested in the logical graph describing the state of the system. As an example, in a physical communication network each node would correspond to router or queue, and a signal may represent the number of packets queuing. However, the state networks we consider here will have one state (node) corresponding to each possible state of the queue.

Complex systems can often be described by a series of state variables such that environmental changes or actions performed by a controller are meant to achieve desired changes to a state variable. For example, these state variables could be discrete, e.g., a counter that registers the number of occurrences of an event and resets to zero. These complex systems are in general non-deterministic, so that for a given action we cannot guarantee that the state of the system will evolve in a predetermined way.

We can create a directed graph where each node represents one possible system state, and directed edges correspond to possible transitions between states. For state variables represented by a counter, each node is associated with a possible counter value and each edge represents possible changes to the counter. If the counter value can only increase or decrease by 1, the corresponding graph will be such that every state will connect only to its two immediate neighbors (corresponding to a ± 1 value for the counter). In principle, we can consider the problem of controlling such a system (selecting actions to be performed at each state) using a graph model. A graph signal for this problem is a "value function" representing a cost or reward associated with each state (node).

Note that since these system are not deterministic, we do not know *a priori* how likely state transitions will be. Then, it may be possible to observe the system to estimate values for those transitions, and corresponding edge weights. However, a major challenge comes from the fact that, in many such systems, the number of possible states is very large. Graph-based formulations have been proposed as a way to address state explosion, as they provide systematic ways to simplify the graphs [5]. Moreover, value functions are often estimated costs or rewards that do not vary significantly from one node to a neighbor, and thus can be viewed as smooth graph signals.

Box 1.6 Finite state machines

- **Nodes**: one per state
- **Edges**: between states for which a transition is possible
- **Degree distribution**: highly irregular
- **Signal**: some metric to quantify a given state
- **Scale**: potentially very large
- **Application**: reinforcement learning, system adaptation

1.4.5 Social Networks

In an online social network each node corresponds to a user, which may in turn correspond to a person or some other entity. Two users are either connected by an edge or not. Thus, there is no natural notion of distance between nodes, edge weights can take only the values zero or one, and these graphs are generally unweighted. In some cases the graph is undirected, if both users have to agree to "friend" each other (see Box 1.7). In other cases, one user may "follow" another user without necessarily being followed back (see Box 1.8). In this case, the graph is directed. Online social networks are communication and information platforms for their users, and a source of valuable information for the companies owning them. We can view any information available on the network as a signal. In some cases, a numerical representation is obvious (e.g., age), while in other cases, such as preferences about a specific topic, it can be constructed in different ways (e.g., it can be assigned to ± 1 depending on whether users prefer cats or dogs, and set to zero if no information is available).

Box 1.7 Online social network – undirected

- **Nodes**: one per user
- **Edges**: between users and their friends
- **Degree distribution**: highly irregular
- **Signal**: different information associated with each user, such as age
- **Scale**: millions to billions of nodes
- **Application**: information mining

In an undirected social network graph (Box 1.7) the connections in the graph may be used to predict information. For example, it may be desirable to poll some users to gather opinions, but impractical to try to poll all users. Then, if users who are connected are more likely to have similar opinions, it may be possible to "sample" carefully (polling only some users) and to then interpolate (infer what connected users may think on the issues in question given the response of their friends). Note that the connections established in a social network do not necessarily entail similar preferences between users. Prediction accuracy will depend on the specific information (signal). Prediction of political preferences probably does not have much in common with prediction of food preferences.

In a directed social graph (Box 1.8), edges link a user to all his or her followers. These graphs can be particularly irregular. Some users may have millions of followers, while others have a handful, something known as a power-law distribution. The graph connectivity has been used to estimate to what degree some users "influence" others. As an example, a graph signal associated with each user could be the number of messages forwarded by followers (e.g., "re-tweets"). Such a signal could be used as a measure of influence. In this case, as in others, more than one graph can be associated with the data. For example, one could consider a graph connecting hashtags and users who have tweets or re-tweets with those hashtags.

Box 1.8 Online social network – directed

- **Nodes**: one per user
- **Edges**: from user to other users he/she follows, and from followers to a user
- **Degree distribution**: highly irregular
- **Signal**: information associated with each user, e.g., geographical location
- **Scale**: millions to billions
- **Application**: identifying influencers

1.5 Mathematical Models of Graphs

In this book we mostly focus on scenarios where signals on a *given* graph are processed. Thus while we will consider what is possible for some specific *classes* of graphs, e.g., bipartite graphs, we will mostly view graphs as being deterministic, rather than random.

Probabilistic graph models In the context of the broad field of *network science* there has been a significant amount of work done in developing probabilistic models of classes of graphs [6]. These models can be used to derive estimates of various graph properties that are valid for those specific graph classes. These models are based on defining the probability that any two nodes will be connected. They are used in order to develop closed form expressions for node degree distributions (the probability that a node has a certain number of neighbors). We briefly describe some of the most popular among these models to illustrate the basic concepts and link specific models to some of

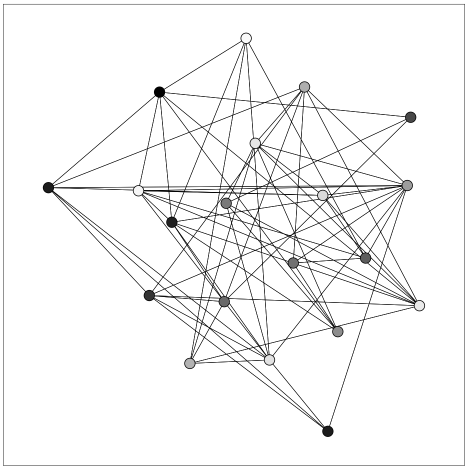


Figure 1.13 Erdős–Rényi graph. These graphs are created by taking every pair of points and connecting them on the basis of a given probability, for the particular graph. Thus the probability gives the proportion of node pairs that are connected. Since the same probability is associated with all pairs of nodes, all nodes will have the same expected degree. Thus, statistically all nodes are the same, even though for a given realization the nodes do not have the same degree (i.e., the graph is not regular).

Code Example 1.2 Code used to generate Figure 1.13. This code takes as a parameter the number of nodes (20), and the probability that any two nodes are connected (0.3)

```
erdos20 = grasp_erdos_renyi(20, 0.3, 'directed', 0);
h = axes;
signal = 255 * rand(1, 20);
grasp_show_graph(h, erdos20,...
                    'node_values', signal, ...
                    'color_map', 'gray',...
                    'value_scale', [0 255]);
```

the examples of graphs discussed so far. Details on how these graphs are generated are given in Section B.2.

The question of how well a specific model fits graphs observed in practice is important, but not one we consider in this book. Probabilistic graph models are useful to evaluate specific algorithms (e.g., graph signal sampling) as they allow us to report results averaged over multiple realizations of a certain type of graph. We can use probabilistic graph models to quantify consistency of performance (variance across realizations) or to identify worst case scenarios in terms of model parameters (rather than specific deterministic graphs).

Erdős–Rényi graphs Erdős–Rényi graphs (Figure 1.13) are defined by a single parameter, the probability that any two nodes are connected. Code Example 1.2 shows the example code used to obtain the graph of Figure 1.13. While this mathematical model allows results about connectivity and degree distribution to be derived, these models may not always capture accurately the structure of many real-world networks. Since every edge has the same probability of being included, all nodes exhibit the same (statistical) behavior, which is rarely the case in practice.

Small-world graphs An alternative model that addresses some limitations of Erdős–Rényi graphs is the Watts–Strogatz or small-world graph (Figure 1.14). These models

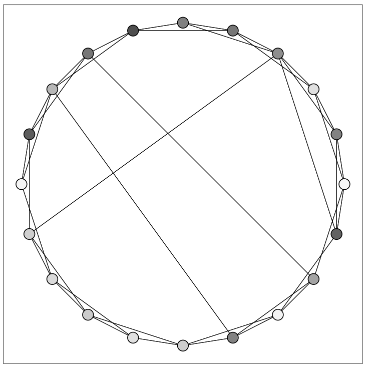
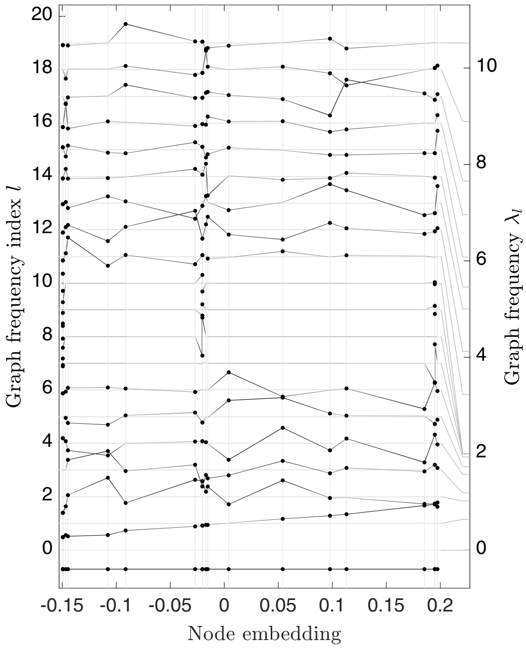


Figure 1.14 Watts–Strogatz or small-world graph. A graph based on this model has the property of being nearly regular (in this example most of the nodes along the circle connect with their immediate neighbors) but also showing random connections across longer distances. This type of graph is designed to capture the behavior of social networks where most connections are local, but a few can go far away, making the average distance between any two nodes in the graph relatively low. This property is popularly known as "six degrees of separation," where any two people are at most six hops apart from each other.

start with a regular graph, such as the cycle graph with N nodes (Figure 1.9). Then, connections are added while ensuring that there are minimal changes in regularity. In the initial model of Figure 1.9, each node has exactly two neighbors. Additional links are included so that each node connects to its k closest neighbors in the original cycle. Finally, with some low probability, edges are added between any two nodes. These added edges provide the small-world property, where it is possible to find relatively short paths between any two nodes.

Barabasi–Albert graphs Barabasi–Albert or scale-free graphs (Figure 1.15) are designed to capture properties observed in social networks (e.g., Twitter) where a few users have orders of magnitude more followers than others (i.e., with some low probability some nodes can have a very high degree). Starting with a connected network, nodes are added so that they connect to a subset of existing nodes, with the probability of connecting to a given node being a function of the degree of the node. Thus, as nodes are added to the network, high-degree nodes are likely to increase their degree even further.

Graph frequencies The example in Figure 1.15 helps to illustrate the idea of graph frequency. This graph has 20 nodes and so we will be able to create a graph signal representation with 20 elementary frequencies; the corresponding eigenvectors are depicted on the left. The definition of these frequencies will be the subject of Chapter 2 and Chapter 3, while the visualization will be described in Section 3.2.6. For now, recall from Figure 1.7 that a signal can be represented as a graph signal and as a 1D signal. This visualization builds on this idea by using an **embedding** (Section 7.5.2) that maps graph nodes into 1D. With this embedding each graph signal can be represented as a 1D signal, as shown on the left of Figure 1.15.



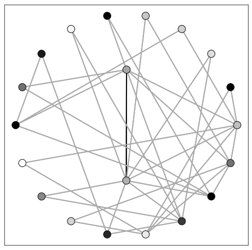


Figure 1.15 Barabasi–Albert or scale-free graph with 20 nodes (right). This type of graph model is meant to reflect the behavior of some real networks where the degree distribution can be modeled by an exponential, that is, it is possible to have nodes with high degree. In this example, the maximum degree is 9, while all nodes have degree at least 2. The graph signal in this example is randomly generated. The plot on the left represents elementary signals associated with the graph frequencies; the vertical displacement of a node represents the value of the signal at the node. Each set of 20 displacements of the 20 nodes constitutes a graph signal representing an elementary graph frequency (an eigenvector of the fundamental graph operator). The left scale (l) and the right scale (λ_l) are the frequency indices and frequency values, respectively, while the lines on the right link the evenly spread elementary graph frequency signals to their corresponding graph frequency (λ_l). This visualization is discussed in Section 3.2.6.

1.6 Roadmap and Approach

The main goal in this book is to provide a basic and intuitive understanding of how to extend to graph signals tools that are standard for regular signals. Chapter 2 introduces basic graph concepts and focuses on graph signal processing in the node domain. Then Chapter 3 introduces graph signal frequencies, building on the tools of spectral graph theory. With these definitions of frequency we can approach classical problems in signal processing such as sampling (Chapter 4) and signal representations (Chapter 5, and also approach more formally the problem of selecting a graph for a specific application (Chapter 6). We then turn our focus to applications (Chapter 7), with examples in several areas including machine learning, sensor networks and imaging. We also provide an overview of basic linear algebra concepts (Appendix A) and introduce in detail the `GraSP Matlab` toolbox for graph signal processing (Appendix B).

One of the challenges in introducing graph signal processing concepts is that often they can be applied to arbitrary graphs, but the same tools may exhibit very different behaviors for each type of graph. In short, the tools are the same but their behavior is very different. We aim to keep discussions general, while also pointing out these differences in behavior through examples. Thus, there is an important caveat. A specific tool, say a graph filtering method, may be applicable to any graph within a given class (e.g., it can be used for any undirected graph) but the results may not be meaningful for some specific graphs within the class (e.g., for random graphs).

Chapter at a Glance

The main goal of this chapter was to motivate the development of graph signal processing tools. We started by introducing some basic graph concepts and explaining what a graph signal is. We then provided some intuition for how a notion of signal variation can be the basis for a definition of graph signal frequency. Next, we gave a series of examples of graphs arising in many different applications, in order to illustrate their broad use and how significantly they can differ in each application. We showed how graphs can be associated with classical signal processing problems (path and grid graphs), and then considered graphs associated with physical networks, information networks, social networks or those arising in machine learning applications. Finally, we provided examples of random graph models, which can be used to generate graphs with given statistical properties.

Further Reading

The study of graphs has been an inspiration for the development of mathematical tools to solve many problems of interest, such as finding the shortest path between two nodes. Spectral graph theory [7, 8] makes use of the eigenstructure of algebraic representations of graphs to estimate graph structure. As an example, we will see that eigenvectors corresponding to low variation can be used to find approximate solutions to the min-cut problem (i.e., finding the partition of a graph into two subgraphs of equal size such that the number of connections or the weights of edges across subgraphs is minimized).

Graph signal processing takes methods in spectral graph theory as its starting point and develops extensions with the goal of providing tools to analyze and represent graph signals. The interest in GSP can be seen as a natural consequence of our increased ability to monitor and sense the environment (e.g., via wireless sensor networks), create complex networked systems (e.g., smart grids or the Internet) and apply optimization to solve large-scale resource allocation problems (e.g., logistical networks). In many of these problems, our goal is not only to understand the graph topology but also to use the graph topology to understand the graph signals (sensor measurements, Internet traffic etc.). As a field, GSP is at a relatively early stage, and one challenge in writing this book has been to decide how much of the recently published work on this topic should be included. Overview papers such as [9, 10, 11] provide perspectives on the topic, while a recently published monograph [12, 13, 14] gives an in-depth treatment of many of the same topics covered in this book.

`GraSP` Examples

Appendix B provides code examples related to specific topics covered in the book. At the end of each chapter, we provide pointers to examples that might be relevant and could be used as the basis for exercises. As a starting point, install `GraSP` (Section B.1). In `GraSP` each graph is represented by a data structure. Random graphs (Section B.2.1) can be generated for the models described in Section 1.5. Using such data structures makes it easy to implement a series of common tasks, such as plotting the graph (Section B.2.1) and associated graph signals (Section B.2.5). While random graph models are useful for developing a better understanding of graph properties, in general it may be necessary to import an existing graph (Section B.2.6). In some cases it may also be useful to build the graph directly with `Matlab` code (Section B.2.7).

Problems

1.1 *Path graph*
Define a length $N = 16$ path graph as in Box 1.1, with a set of positive edge weights a_{ij} between consecutive nodes. Use the variations TV_t and TV_g defined in (1.2) and (1.3), respectively.

a. Give a definition of the term “node” and a choice of a_{ij} for which the two variations are the same.

b. For the variation (1.2), assuming the points are equally spaced, find a signal x such that: (i) $\sum_k |x(k)| = N$, (ii) $\sum_k x(k) = 0$ and (iii) TV_t is minimal.

c. Assuming that $k_9 - k_8 = 1 - \epsilon$, $\epsilon < 1$, and the remaining intervals remain the same, find the new signal with minimal variation under the conditions in part **a**. Compare the two solutions and discuss.

d. What happens as ϵ in part **c** increases? Relate the change in edge weights to the change in the signal.

1.2 *Images as graphs*
Define a graph associated with an image (Box 1.2), with each pixel connected to its four neighbors (except at image boundaries) and with all edge weights equal to 1. For connected pixels i and j, denote as $\delta_{ij} = a_{ij}|x(i) - x(j)|$ the term in the summation of (1.3).

a. Write `Matlab` code to load an image and produce another image where each pixel corresponds to one of the δ_{ij} values (i.e., convert each edge in the original graph into a pixel in the resulting image). Discuss how the pixels in the output image depend on the characteristics of the pixels in the original image (e.g., by comparing smooth regions to texture regions in the original image).

b. Repeat part **a** but using, for the same pairs of connected edges, a new weight that depends on pixel intensities:

$$a_{ij} = \exp \frac{-(x(i) - x(j))^2}{\sigma^2}.$$

How do the results change? How does the choice of σ affect the output images?

1.3 *Erdős–Rényi graphs*

Use Code Example 1.2 to generate Erdős–Rényi graphs with different probabilities p. Notice that for lower probabilities it is more likely that the graph will become disconnected.

a. Write a `Matlab` code to determine whether the graph is disconnected.

b. Use simulations to generate an empirical estimate for the probability of a graph being disconnected (for a given probability p).

1.4 *Watts–Strogatz graphs*

Repeat Problem 1.3 for the Watts–Strogatz model of Section B.2.1.

1.5 *Exploring social network graphs*

This problem can be completed with data from any social network, either directed (Box 1.8) or undirected (Box 1.7). If the number of connections involved is large, use sampling to simplify the process, i.e., select a random subset of neighboring nodes and use only information from that subset.

a. Data collection: for your chosen social network, select one user, *Alice* (this could be your account or another account for which information is accessible) and record the following information: (i) the number of connections *Alice* has, (ii) the number of connections of each of *Alice*'s one-hop neighbors and (iii) the number of shared connections between *Alice* and her connections (if *Alice* and *Bob* are connected, how many people connect to both *Alice* and *Bob*).

b. Discussion: this is an open ended question with the goal of describing the structure of these social network graphs. Here are example questions that could be considered. Is there a lot of variability in (ii) across your neighbors? What are the maximum and minimum variabilities? How do you interpret this variability? What do the shared connections of (iii) tell you about the existence of communities?

1.6 *Random geometric graphs*

Generate a random geometric graph (see Section B.2.3) and note that graph nodes can be interpreted as randomly located sensors in space. Denoting by `geom` the generated graph, `geom.layout` gives the position of the nodes in space, while `geom.A` is the adjacency matrix (to be introduced in Chapter 2). In this symmetric matrix, the entry a_{ij} is equal to the inverse of the distance between node i and node j, $1/d(i, j)$. Typically a threshold is applied so that a_{ij} is set to zero if $1/d(i, j)$ is small. Thus, you can assume that nodes for which $a_{ij} = 0$ are not connected.

a. Write `Matlab` code to compute (1.3) for `geom` and use it to find the variation for a random vector $\mathbf{x}$.

b. Most sensor networks observe the data generated independently of the sensors. To simulate this, generate a high-resolution random image using `Matlab` and assume this image represents a regular sampling of the same region as that represented by `geom`. Then, create a graph signal **x** by assigning to each node location in `geom.layout` the intensity of the closest pixel to that location in the image you generated.

c. Perform simulations to show how the variation computed in part **a** changes when the images used to generate the graph signal in part **b** have different properties (e.g., smooth images as against noisy images). Discuss.

d. In part **c** we considered a fixed set of sensors and changed the images used to generate the graph signal **x**. Now do the opposite: fix the image to generate graph signals, but compare the variation obtained when different random geometric locations for the sensors are selected. Is the variation consistent across multiple graph realizations? Discuss.