1 Graphs and Graph Signals

This book provides an introduction to graph signal processing (GSP), also known as signal processing on graphs. In this chapter we introduce several applications where graphs are useful, as a way to highlight the broad diversity of graph types. We will call a "graph signal" data associated to each of the nodes of a graph. As an example, a graph can be constructed with nodes corresponding to a set of sensors, and a graph signal comprising measurements obtained from the sensors.

One of the most appealing, but challenging, characteristics of graph signal processing is that it provides a common framework to study systems that differ very significantly in their behavior and properties. This diversity comes primarily from the size and topology of the corresponding graphs, and will be illustrated throughout this chapter. Thus, while we will be able to define a common set of tools that work with any graph, the interpretation of the methods and the insights derived from their use are potentially quite different from case to case, as a function of the graph characteristics. 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 this chapter we provide examples of various scenarios where graphs can be used to model systems of interest, and to use this as motivation to explore graph signals and graph signal processing. The graphs corresponding to some of these motivating scenarios will be used as examples throughout the text.

1.1 Graphs

Graphs have long been used to represent information about objects (concepts, locations, etc) and their relationships. There are many examples, some of which will be discussed below, that can be associated to different kinds of relationships, such as "A is at certain distance of B", "B may happen if A happened", or "A and B are similar". Graph representations are general, thus providing a common language and common mathematical tools for very different problems and applications.

We define graphs in terms of a *set of vertices or nodes*, $\mathcal{V} = \{v_1, v_2, \dots v_N\}$ or

Draft version -- Compiled on 2018-03-30 11:24:15-07:00

$\mathcal{V} = \{1, 2, \ldots, N\}$[1] and a *set of edges* $\mathcal{E} = \{e_1, e_2, \ldots e_M\}$ or $\mathcal{E} = \{1, 2, \ldots, M\}$, where we may also denote an edge as $e = v_i v_j$ if it connects vertices i and j.

Graphs can be *directed* (one vertex may be connected with another, but not the other way around) or *undirected* (when two vertices that are linked are always connected to each other). The edges can be *weighted* or *unweighted*. In this book, when discussing theoretical concepts we will focus on general graphs, that is, we will not make any assumptions about the number of vertices, the number of edges per vertex, whether the graph is regular (i.e., all vertices have the same degree), etc. Specific examples will serve to provide insights about concepts, and also to show the different types of graphs encountered in practice.

Each of the nodes in a graph may be associated to a scalar value. Generalization to the case where vectors are associated to each vertex is not explicitly considered here. The aggregation of all these values into a vector of dimension $|\mathcal{V}| = N$ will be what we call a *graph signal*. For a given graph, there can be many different graph signals, e.g., a sensor network associated with a graph representing the relative positions of the sensors may have graph signals representing temperature or humidity measurements by each sensor. Similarly, the same set of sensors can capture measurements at different times, leading to a time-varying graph signal. We will be interested in analyzing these graph signals, which will lead to different interpretations depending on the topology of the graph.

We will describe methods that lead to frequency representations for these graph signals, and therefore will allow us to analyze, filter, transform and sample them in a meaningful way that takes into account signal variation on the graph. In terms that would be familiar to a signal processing practitioner, we want the tools that will allow us talk about "low pass" graph signals, with corresponding "low pass" filters. As for the sampling problem, we would like to be able to define the required level of "smoothness" a signal is required to have in order for it to be recovered from a set of signal samples.

There are many problems of practical interest in which a graph-based approaches can be pursued. In some cases the choice of graph is obvious given the application, while in others choosing the "right" graph is key in achieving meaningful results. We will discuss both types of scenarios, including techniques to optimize the graph selection.

As we start providing examples of problems with corresponding graph representations, it is important to keep in mind that **different graphs can be chosen to analyze a given dataset**. This is an important choice to make, as the choice of graph parameters affects the results. Moreover, we will be able to develop **frequency interpretations for any graph**, but these interpretations will not be unique.

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

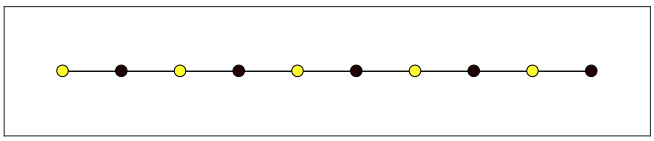

Figure 1.1 Simple path graph

1.2 Examples

We start by presenting 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 what may be a signal of interest. We also include in our discussion the **degree distribution**, i.e., how the number of edges or their weight of a node (the degree) varies across the graph.

1.2.1 Graphs and classical signal processing

We start by introducing two graphs closely linked to familiar signal processing problems, namely, line graphs and grid graphs.

Line graphs and discrete-time signals
Discrete (and finite length) time signals can be interpreted as line graphs where each sample in time corresponds to a node, and the edges between nodes all have weight equal to 1. Choosing equal weights make sense if time samples are equally spaced in time.

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)
- Signal: values of each sample in time

Alternatively, we could view such a signal as one period of an infinite length periodic signal with period N. This signal can then be associated to a circular graph with N vertices. In the first case the graph is almost regular, i.e., same connectivity for each node except the two end nodes, while in the second case (circular) the graph is exactly regular. In the circular graph case, as will be seen later, the transform associated to the graph will be the Discrete Fourier Transform (DFT).

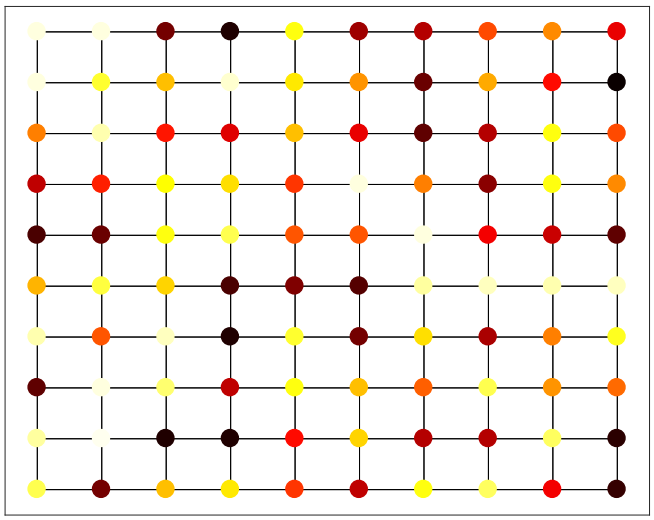

Figure 1.2 Simple grid graph

Digital images

2D Images can be interpreted in a similar way by defining a regular graph (e.g., one where each pixel is connected to its 4 immediate neighbors) except at the boundaries. Of course we could make the graph exactly regular by connecting it in a circular way similar to what would be done in the cases of a line graph. This can be done by connecting the left and right nodes of each horizontal line graph, and similarly the top and bottom nodes of each vertical line graph. Assume that pixels are indexed by $x(i,j)$ where $i,j = 0,\ldots,N-1$ then in a 4-connected graph a given pixel would be connected to 4 neighbors: $x((i+1) \mod N, j)$, $x(i,(j+1) \mod N)$, $x((i-1) \mod N, j)$, and $x(i,(j-1) \mod N)$, where the modulo operation wraps the connection. For example, the pixel at the top right corner will have as a neighbor to its "right" the pixel top left corner, and its neighbor "above" will be the bottom right pixel. The 2D separable DFT will be associated to that graph.

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

While signals defined on the graphs of Boxes 1.1 and 1.2 can already be ana-

lyzed using existing signal processing methods, these examples are still important for several reasons:

- For certain choices of these graphs, tools from graph signal processing exactly corresponding to those developed in classical signal processing, thus allowing us to view graph signal processing tools as "natural" generalization of existing methods. As example, when all edge weights are equal to one in the path graph, the graph frequencies correspond exactly to the discrete cosine transform (DCT) bases.
- Second, as graphs deviate from the properties of those matching conventional signal processing (e.g., when edge weights are no longer equal), we can observe how the corresponding signal representations change, which provides insights about how the tools we develop adapt to changes in graph topology.
- Finally, we will be able to observe that the closer graphs are to being regular the more the tools associated to them will behave as expected based on our knowledge of signal processing. As an example, in conventional signal processing we expect that exact localization in time and space is not possible, but we shall see that exact localization is indeed possible for graphs.

1.2.2 Physical networks

As technology for sensing, computing and communicating continues to improve, we all 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.

The Web

The Web can be easily seen as a graph, where each page corresponds to a vertex, and the graph is directed, since a page can link to another, without the linking being reciprocal. This graph structure has been often 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 vertex (blog) by creating a histogram of the frequency of appearance of certain keywords in that blog.

Note that graph-based representations of the web were at the core of 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.

Box 1.3 The Web

- Nodes: one per webpage
- Edges: between webpages that reference each other
- Degree distribution: highly irregular
- Signal: some metric to characterize a particular webpage or blog (e.g., a distribution of frequency of occurrence of certain keywords)

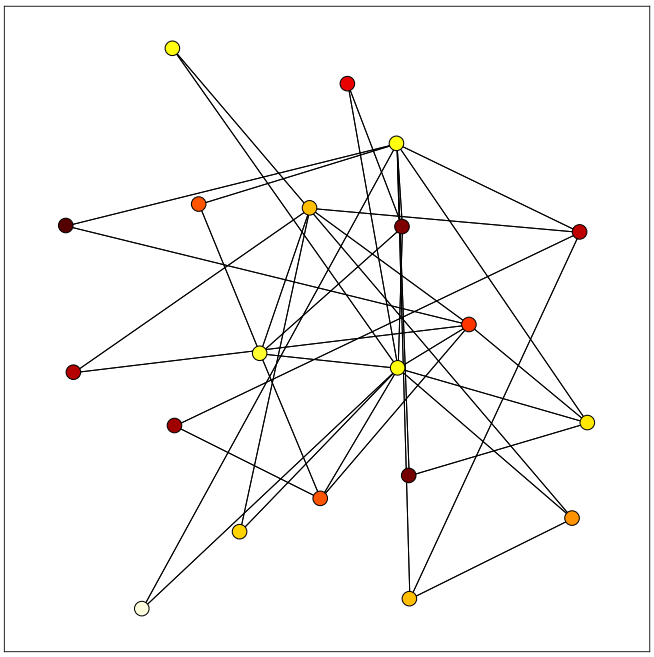

Figure 1.3 Barabasi Albert or Scale-free graph

Sensor Networks

A variety of systems can be described as a sensor network. Examples include sets of distributed temperature sensors (indoors or outdoors), surveillance cameras or even devices, such as cellphones, carried by a number of users.

A sensor network may be deployed to sample information from the real world, e.g., temperature. Thus, the nodes are likely to have a location in space, so that edge weights between nodes can be made a function of the distance between the corresponding sensors, and may even be time-varying. The example of Fig. 1.4 shows a regional network of weather sensors. Sensors can be deployed at much smaller scales, within a building or on a bridge for example, and measure temperature, air quality, vibrations, etc. As a clear consequence of Moore's Law, our ability to sense, record information, store it and transmit it has continued to

Box 1.4 Sensor Network

- Vertices: 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, k-nearest neighbor graph
- Signal: sensor measurement, for example temperature

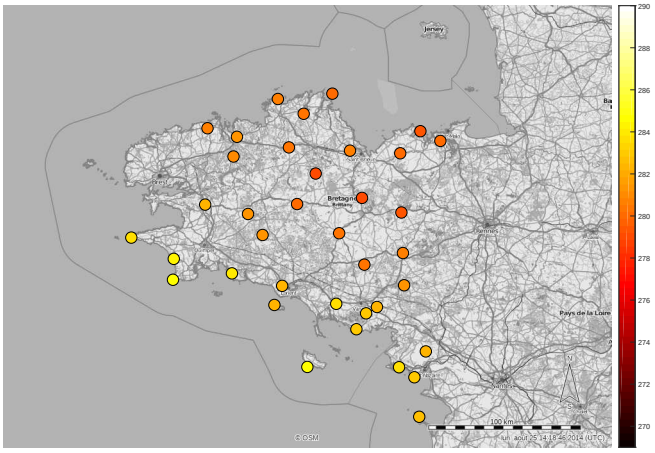

Figure 1.4 Sensor network

increase, so that sensor networks, now considered components of the Internet of Things, are an increasing part of our everyday life. A main objective of GSP is to develop tools to analyze the large amounts of data captured by these sensing devices.

1.2.3 Learning with graphs

Classification is a typical machine learning task, which involves developing algorithms that can associate a label to data. For example, we may have a collection of images belonging to some categories (dogs vs cats, say) and we would like to automate the process of determining to which category a new image belongs to. 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 to 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 vertex corresponds to a data point, i.e., one of the images in the training set, and the edge weights are a function of the similarity between two

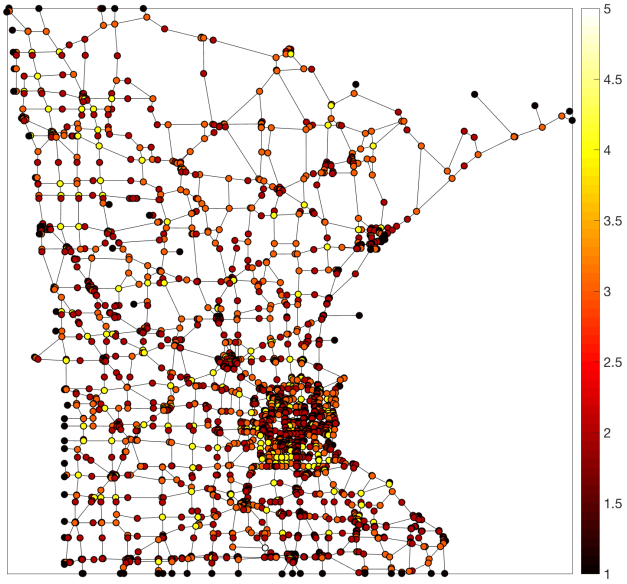

Figure 1.5 Minnesota road network graph

data points, i.e., how similar the two images are the chosen feature space. In a similarity graph with normalized edge weights (maximum value equal to 1), two nodes that correspond to similar images will have an edge with weight close to 1 between them. In contrast, two nodes corresponding to two different categories will not be connected, or the edge between them will have weight close to zero.

Box 1.5 Learning: Similarity Graph

- Nodes: one per data point used in the learning process
- Edges: between data point as a function of distance in feature space
- Degree distribution: can be regular, e.g., k-nearest neighbor graphs
- Signal: label for each data point

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 that when choosing a random node its immediate neighbors on the graph will be likely to have the same label. We will

view this as a "smoothness" associated to the label signal, will introduce this notion more formally and apply it to learning in Chapter 8.

1.2.4 Analyzing Complex systems

Finite State Machines

Many systems of interest can be described by a state variable, such that environmental changes or actions performed by a controller will lead to a change in the value of the state variable. If the state variables can only take discrete values, e.g., a counter that registers the number of occurrences of an event and resets to zero, we can create a directed graph that captures all the possible states of the system as well as allowable transitions. We can then consider the problem of controlling such a system (selecting actions to be performed at each state) based on a graph formulation, by associating a "value function" to each node of the graph (state of the finite state machine).

Box 1.6 Finite state machine

- 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

1.2.5 Online social networks

Friendship Graphs

In this case there is no notion of distance between vertices. Instead, two nodes may or may not be connected. If they are the edge has weight one. The graph is undirected since both users agree to "friend" each other.

Box 1.7 Online Social Network – Undirected

- Nodes: one per user
- Edges: between a user and his/her friends
- Degree distribution: Highly irregular
- Signal: different information associated to each user, such as age, for example

In this scenario, or in other situations where the goal is to analyze data available from a social network, it may be useful to determine if the connections in the graph allow us to predict information. For example, it may be desirable to

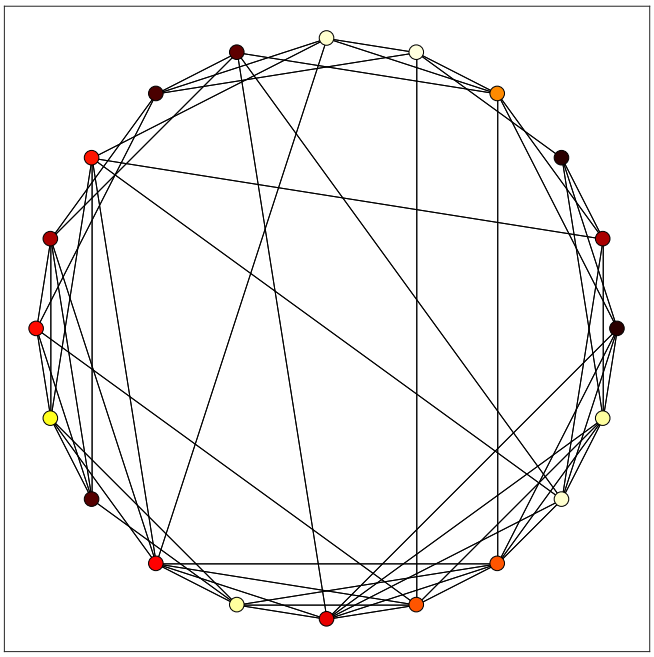

Figure 1.6 Watts-Strogatz or Scale-free graph

poll users to gather opinions, but impractical to try to poll everybody. Then, if users who are connected are more likely to have similar opinions, it may be possible to "sample" carefully (polling only some users) to then "interpolate" (infer what connected users may think on those issues given what their friends responded).

Online Social Networks – Twitter

Here the graph is directed and is generated by linking 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, by for example signals such as the number of messages forwarded ("re-tweets"). For example, one could consider a graph signal of the number of times a message from a given user has been retweeted by each of its direct and indirect followers as a measure of influence. In this case as in others more than one graph can be associated to the data. For example, one could consider the graph connecting hashtags and users who have tweets or re-tweets with those hashtags.

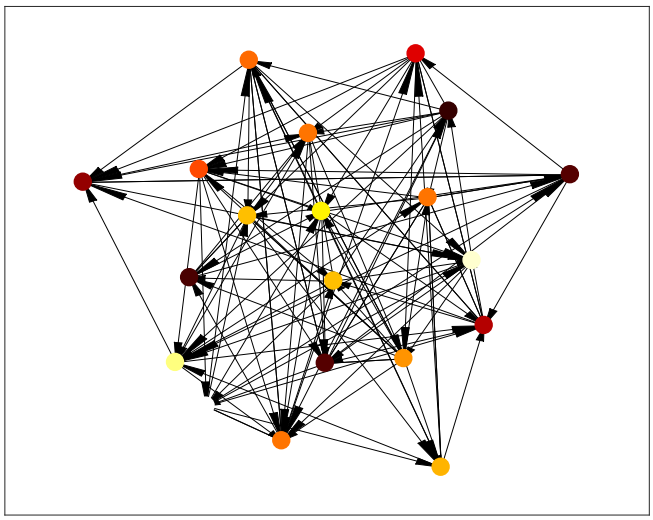

Figure 1.7 Erdos-Renyi Graph

Box 1.8 Online Social Network – Directed

- Vertices: 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 to each user, e.g., geographical location.

1.3 Signals vs Graph Signals

For any of these and other examples, we can define the graph signal as the information associated to each of the vertices. Denote $f(v) \in \mathbb{R}$ the scalar signal at vertex $v \in \mathcal{V}$. Except where otherwise indicated we will focus on signals that take a real scalar value at each vertex v. This could be generalized to scenario where the signal at each vertex is complex, a vector, or a time series.

Note that when we consider graphs with N nodes we are essentially working with vectors in $\mathbb{R}^N$, so that if we combine all the $f(v)$ into a single vector $\mathbf{f} \in \mathbb{R}^N$ we would have the same information. The key question we must be asking then is: *why not operate directly with* $\mathbf{f}$ *and use standard methods from linear algebra or transforms such as the Discrete Fourier Transform (DFT) for* $\mathbb{R}^N$*?*

First, note that with unweighted graphs associated to regular domain signals (e.g., a line graph with edge weights all equal to 1 as discussed earlier), one could use existing transforms to analyze the signal. However, if the graph maintains the same topology but has weighted edges, existing transforms no longer take

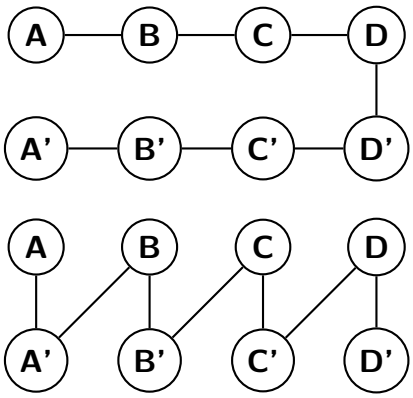

Figure 1.8 Graph $\mathcal{G}$

Figure 1.9 Graph $\mathcal{G}'$

into account the different "relation" between samples captured by these unequal weights.

For regular domain signals the sample indices are meaningful, e.g., in the time domain sample $x(n+1)$ comes after sample $x(n)$. For graph signals, the indices associated to each vertex are arbitrary. Thus, for two nodes i and j, the values i and j do not matter, what matters is whether there is an edge between i and j. We can change the labels and still preserve the connectivity. This will be a simple permutation of the adjacency matrix, as will be seen later. 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 1D vector.

Second, the same signal may have very different interpretations depending on how the vertices are connected. This is really the main motivation for graph signal processing and can be illustrated with a simple example.

Example 1.1 Consider the two graphs $\mathcal{G}$ and $\mathcal{G}'$ (see Figures 1.8 and 1.9) and assume the same graph signal is associated to both of these graphs, for example, a signal $\mathbf{f}$ defined as

$$\begin{aligned} f(A) = f(B) = f(C) = f(D) = 1 \\ f(A') = f(B') = f(C') = f(D') = -1. \end{aligned} \tag{1.1}$$

We can easily see that when $\mathbf{f}$ is associated with $\mathcal{G}$ it exhibits much less "variation" along the graph as compared to when it is associated with $\mathcal{G}'$. As one traverses $\mathcal{G}$ node by node from A to A', $\mathbf{f}$ changes sign just once, between D and D', whereas the sign changes from vertex to vertex when we traverse $\mathcal{G}'$ from A to D'. We will later see that this intuition about variability within a graph can be approached more formally and related to similar notions used for signals in regular domains. [50]

Example 1.3 also raises an important point: in *many Graph Signal Processing problems it will be necessary to choose the graph that is best suited for the specific task.*

As an example of how different connectivity can be incorporated into our signal analysis, consider a social network, which we assume will collect, or infer, relevant user information (geographical location, age, occupation, perhaps even income range). Analyzing this information taking into account the connectivity of the graph would allow us to answer questions about how users relate to each other. For example, if users of similar ages are likely to be "friends" in a given social network, this could be noted by observing that the age signal is smooth on the social network graph.

Similar kinds of assessments could be made about more complex signals, such as information about user interests that could be derived from postings[2] but the insights will be the same. As an example, one could define a vector where each entry represents a topic, and the corresponding numerical value is associated to an estimated level of interest a user has on that topic. Essentially by observing the variations across edges in the graph we will be able to infer characteristics of datasets that would be difficult to visualize, given the complex connectivity of the graphs. This can be useful for analysis (is the graph signal varying smoothly?), as well as for sampling and interpolation (what is a sufficient number of measurements to capture in order to have to be able to reconstruct the signal from the samples), denoising, etc.

In sensor networks samples are irregularly obtained within a spatial region, and can be related to each other based on different graphs, with edge weights and connectivity based on distance, radio connectivity, or some other side information.

Here also, knowing the connectivity of the graph allows to process the corresponding graph signal in different ways. For example, with a graph based on distance could be used in order to determine the best way to sample information (say, temperature) across a set of sensors, knowing that it is likely to vary slowly (smoothly) across the distance-based graph.

1.4 Roadmap

Our goal in this book is to provide a basic and intuitive understanding of how it is possible to extend to graph signals tools that are standard for regular signals such as:

- Filtering
- Frequency domain representations and their interpretation
- Transforms/Wavelets

[2] For example, a bipartite graph where some vertices represent topics and others represent users, with edges present if a user is interest in a topic

- Sampling
- Interpolation

and to illustrate the basic ideas with examples in a diverse range of applications.

Chapter at a glance

The main goal of this chapter was to provide a series of examples of graphs arising in many different applications. In particular we considered graphs associated to

- classical signal processing problems (path and grid graphs)
- Physical networks (sensor networks or the web)
- Learning problems (similarity graph)
- Online social networks

Further Reading

The study of graphs has been an inspiration for the development of mathematical tools to solve many problems of interest, such as for example finding the shortest path between two nodes. The tools of **Spectral Graph Theory** [1] make 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 used to find approximate solutions to the min-cut problem (i.e., finding the partition of a graph into two sub-graphs of equal size such that the number of connection or weight of edges across sub-graphs is minimum.)

Graph signal processing builds on top of existing tools by using the eigen-decompositions developed by spectral graph theory, and others developed as extensions, in order to provide tools to analyze and represent graph signals. Interest in these tools 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, Graph Signal Processing is at a relatively early stage, and one of the challenges in writing these notes has been to decide how much of the recently published work on this topic should be included. Recent overview papers [2][3][4] provide perspectives on the topic.

Problems

1.1 Consider the 1D signal graphs of Box 1.1. Assume the edges of these graphs can chosen to have different weights? What could these weights represent?

1.2 What would be the effect of including weights in the case of Box 1.2 As we shall see, weights can encode some similarity between pixels, so that larger weights indicate stronger similarity.