1 Graphs and Graph Signals

Graph signal processing (GSP), or signal processing on graphs, focuses on methods to analyze and process data associated to graphs. A graph consists of a series nodes, whose relations are captured by edges. Two nodes are connected when there exists an edge between them. This connection may represent an actual physical connection (e.g., two nodes in a communication network connected by a physical communication link) or some other property (e.g., two people connected in a social network). In some cases, a (positive) weight is associated to an edge so as to capture how strongly connected the two nodes are (larger weights indicating stronger links). Then, a "graph signal" is a vector representing data, with each entry corresponding to data value associated to one of the nodes.

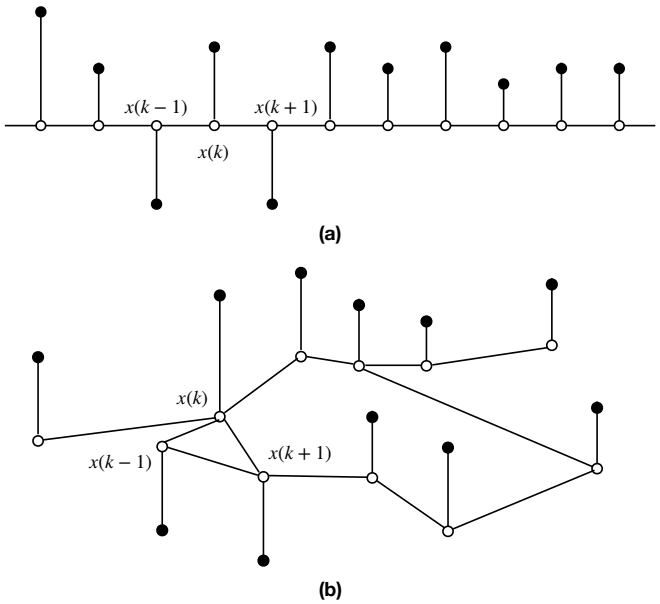

Figure 1.1 Comparison of notions of variation and frequency in time and vertex domain. The same signal is seen as (a) observations in time and (b) placed on a graph. The signal in (a) can be viewed as being positioned on a line graph, while in (b) we have an arbitrary graph.

Draft version -- Compiled on 2018-07-02 02:29:01-07:00

Conventional signal processing is concerned with the development of models and processing tools for signals that are defined in time (e.g., speech, audio) or in space (e.g., images). A key concept in the development of signal processing tools is that of **frequency**. An intuition about the frequency of a signal can be reached by observing how quickly a signal changes from sample to sample. This is illustrated by the example in Figure 1. Figure 1(a) represents a signal in time. In order to evaluate its variation we can look at consecutive samples along time. Notice that $x(k)$ is quite different from both $x(k+1)$ and $x(k-1)$, and so we can see that the local variation, and thus the local frequency, are high. In other parts of this time signal variation is not as fast, and so the local frequency is lower. This book will develop similar insights for graph signals, where rather than consecutive samples in time, we will consider how much variation there is the neighborhood of a node. Referring to Figure 1(b), we can see that nodes k, $k-1$, $k+1$ have the signal values, but now the connection between these nodes and others are not so easy to explain. For example, node k is connected to two other nodes, aside from $k-1$ and $k+1$, and since these nodes have values closer to $x(k)$ the frequency of the signal on the graph of Figure 1(b) may be lower.

While definitions vary depending on the type of signal being considered (e.g., continuous time vs discrete time, 1D vs 2D, etc), the main steps followed are very similar:

- Define a series of elementary functions (bases) each of them having a different frequency interpretation
- Develop tools so that each signal can be represented in a unique way based on its frequency components.
- This will allow us to develop a language to characterize observed signals (e.g., low frequency or smooth signals, who show a slow variation from node to node).
- Design filters that can remove some of the frequencies from a specific signal (e.g., remove higher frequencies in order to "denoise" the signal).

Example 1.1 Sensor network graph. Consider a set of sensors measuring temperature inside a large factory. The goal is to analyze temperature measurements so as to determine whether there may be problems with some of the equipment. As an example, if some of the equipment is overheating this could be observed through locally higher measurements. Discuss how a graph could be constructed in order to achieve this goal, by associating one sensor (and sensor measurement) to each node of the graph.

Solution

In this type of application, the goal is to detect an "anomaly", by comparing temperature in different parts of the factory floor, and detecting unexpected behavior. As an example, this could be done by comparing each temperature

observation made by a sensor, to those made by other sensors nearby. The intuition behind this is that we may not find surprising to measure very different temperature values in areas far away, but if two sensors are nearby and temperature is much higher in one, this could be an indication of equipment (or sensor) malfunction. Thus, we can consider a graph where edges having weights that are decreasing functions of the distance, and where only nodes close enough to each other will be connected. A very simple detection strategy may then involve comparing values (signal) at a node, to those in the immediate neighborhood.

One of the most appealing characteristics of graph signal processing is that it provides a common framework to study systems that are fundamentally different in their behavior and properties. The size of the graphs can vary significantly, from a few hundreds of nodes (e.g., in a sensor network) to millions (e.g., an online social network). Similarly, graph topologies can be very regular (any node has the same number of connections) or highly irregular (some nodes have orders of magnitude more connections than others).

Having a single set of mathematical tools to describe many different types of graphs is appealing but also challenging: while the tools are the same, 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.

This chapter provides examples of various scenarios where graphs can be used to model systems of interest and uses these as motivation to explore graph signals and graph signal processing. Some of these motivating scenarios will be used as examples throughout the text.

1.1 Graphs and graph signals

1.1.1 Basic concepts

Graphs representations have long been used to provide information about objects (concepts, locations, etc) and describe their relationships. Connections between two nodes, A and B, can be very different in nature depending on the application at hand. For example a connection could be used to describe that "A is at certain distance of B" or that "B may happen if A happened" or simply that "A and B are similar". Graph representations are general. They provide a common language and associated mathematical tools for very different problems and applications.

More formally, we define graphs in terms of a *set of vertices or nodes*, $\mathcal{V} =$

$\{v_1, v_2, \dots v_N\}$ or $\mathcal{V} = \{1, 2, \dots, N\}$[1] and a *set of edges* $\mathcal{E} = \{e_1, e_2, \dots e_M\}$ or $\mathcal{E} = \{1, 2, \dots, M\}$. We may also denote an edge as $e = v_i v_j$ if it connects vertices i and j. The edges can be *weighted* or *unweighted*. Some graphs are *directed*: $v_i v_j$ may exist, while $v_j v_i$ does not. Others are *undirected*: $v_i v_j$ and $v_j v_i$ both exist, and if the graph is weighted both edges have the same weight.

In this book, theoretical concepts are introduced for 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 then serve to provide insights about concepts, and also to show the different types of graphs encountered in practice.

Each node i in a graph has a scalar value $f(i)$ associated to it. The aggregation of all these values into a vector of dimension $|\mathcal{V}| = N$, $\mathbf{f}$, will be called a *graph signal*. Generalization to the case where vectors are associated to each node is possible, but not explicitly considered here. For a given graph, there can be many different graph signals.

Example 1.2 In the sensor network of Example 1.1, the same graph could have both temperature and humidity signals. Similarly, the same set of sensors can capture measurements at different times, leading to a time-varying graph signal.

Graph signal processing methods can be viewed in the context of data science applications, where the goal is to extract useful information form data. To that end, GSP methods lead to processing and analysis of data that takes into account the topology of the graph. In other words, a given graph signal $\mathbf{f}$ can be interpreted in different ways depending on the graph it is associated with.

Example 1.3 In Example 1.1 assume that sensors are deployed inside a building. Then we will get a different analysis of the signal depending on whether the graph is defined based on physical distances between sensors only or, instead, weights take into account the presence of walls separating some sensor locations.

The key insight that GSP provides is based on capturing the variation of signals across connected nodes.

Example 1.4 In Example 1.1, if two nodes i and j have very different temperatures, this will mean different things depending on how strongly connected they are. If the two nodes are strongly connected (e.g., they are close together), a

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

big difference between $f(i)$ and $f(j)$ could be a clue to detect some unexpected behavior (an overheating device in a room, say), whereas if the two nodes are not connected, the prior information provided by the graph tells us that we should not necessarily expect the temperature values to be similar (e.g., one sensor is indoors, the other is outdoors).

This idea of quantifying the variation of signals across nodes will lead us to introduce frequency representations for graph signals, where high frequency will capture 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 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.

Example 1.5 Considering again Example 1.1, a low pass filter could be used to reduce noise in the temperature measurements before processing them, while sampling would allow us to decide which subset of sensors should be activated to measure, in case we would increase battery life by not having all of them active simultaneously.

There are many problems of practical interest in which 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.

1.1.2 Signals and 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 can combine all the $f(v)$ into a single vector $\mathbf{f} \in \mathbb{R}^N$, so that we are essentially working with vectors in $\mathbb{R}^N$. Thus it is worth asking why we could not simply work with $\mathbf{f}$ as a vector in $\mathbb{R}^N$ and apply existing methods from linear algebra to transform this vector.

First, 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 1.6 Consider the two graphs $\mathcal{G}$ and $\mathcal{G}'$ in Figures 1.2(a) and (b). The same graph signal is associated to both graphs. For which of those two graphs does the signal have higher variation?

Solution

Comparing Figures 1.2(a) and (b), we can observe that there are only two nodes that negative values. Note that those two nodes have have only one connection to others with positive values in Figure 1.2(a), while several such connection exist in Figure 1.2(b). From this we may infer that the signal has higher frequency for the graph of Figure 1.2(b).

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.

Example 1.6 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,

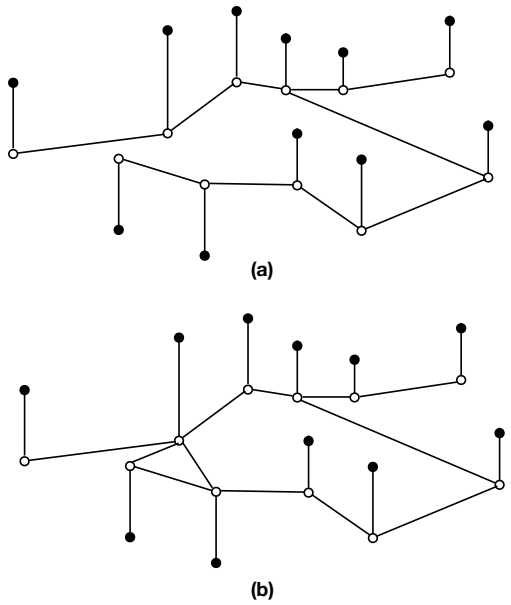

Figure 1.2 The same graph signal is shown associated to two different graphs. Note that how the nodes are connected affects how much variation we observe for the same signal. For example, in (a) there are fewer connections between nodes having values with opposite sign than in (b). This will be shown to indicate that the signal has higher frequency in the graph of (b).

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 num-

[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

ber 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, 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.

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**, i.e., how the number of edges or their weight of a node (the degree) varies across the graph. Examples in this section are generated with Matlab using the GraSP toolbox, which will be used throughout the book and is introduced in more detail in Appendix A. Code used to generate many of the figures in the book is available from the book webpage.

1.2 Graphs and classical signal processing

We start by introducing two graphs closely linked to familiar signal processing problems, namely, line graphs and grid graphs. Frequency representations for these graphs match exactly those available for some specific signals studied in classical signal processing.

Line graphs, cycle graphs and discrete-time signals

Discrete (and finite length) time signals can be interpreted as graph signals, where each sample in time corresponds to a node, and the edges between two neighboring nodes (corresponding to consecutive samples in time) have weight equal to 1. Choosing equal weights seems to be a good choice if time samples are equally spaced in time.

One possible graph choice for this type of signal would be a line graph such as that in Figure 1.3, where all edge weights are equal to 1 and nodes are ordered in time. The graph frequency definitions we will develop in Chapter 4 can be shown to correspond exactly to the discrete cosine transform (DCT), a popular tool for analyzing and representing signals.

Alternatively, we could associate the same signal to a circular graph as the one shown in Figure 1.4. This would be equivalent to viewing such a signal as one period of an infinite length periodic signal with period N, the number of nodes.

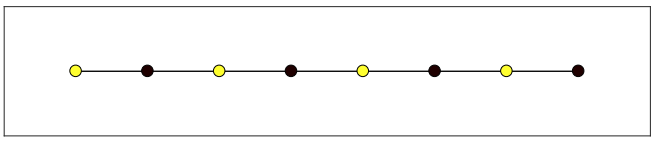

Figure 1.3 Line graph to represent signals in time as graph signals. If all edge weights are equal to 1 and nodes ordered in time, the frequency representation for these graph signals will match those of discrete final length time signals.

Listing 1.1 Matlab code used to generate Figure 1.3

```
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', 'hot',...
                 'value_scale', [0 255]);
ylim([4 6]);

save_figure('path8');
```

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 in Chapter 4, the transform associated to the graph will be the Discrete Fourier Transform (DFT).

Comparing the two representations of the same signal in Figs. 1.3 and 1.4 provides a concrete example of how the same signal can be associated to different graphs, leading to different interpretations. When the line graph of Fig. 1.3 is used, the two end points are not connected so that if they have very different values this will have no effect on the smoothness of the signal. Conversely, when the cycle graph of Figure 1.4 is used, those two end points will be connected (corresponding to the connection of consecutive cycles in a periodic signal) and a difference in value between those two nodes will lead to higher variation (less smoothness) in the graph signal. Comparing these two graph choices also reminds us that conventional signal processing also relies on having multiple different representations for the same signal (e.g., DCT and DFT, among many others), with the choice to be made dependent on the specific application.

Digital images

2D images can also be interpreted as graph signals in a similar way by defining a regular grid graph (e.g., one where each pixel is connected to its 4 immediate neighbors) except at the boundaries, as shown in Figure 1.2.

As in the case of line graphs, we can also create a graph that is exactly regular by connecting row line graphs as cycles, so that each row line graph (similar to Figure 1.3) becomes connected as a cycle (similar to Figure 1.4). The same

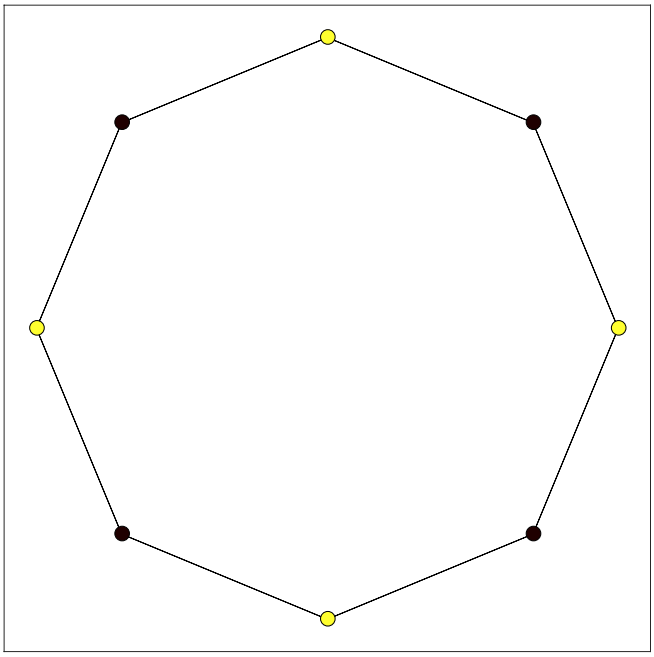

Figure 1.4 Cycle graph with the same graph signal as in Fig. 1.3

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: values of each sample in time
- Scale: each finite set of samples can be viewed as a "window" for time signal analysis

can be done for line graphs corresponding to columns. Thus we connect 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

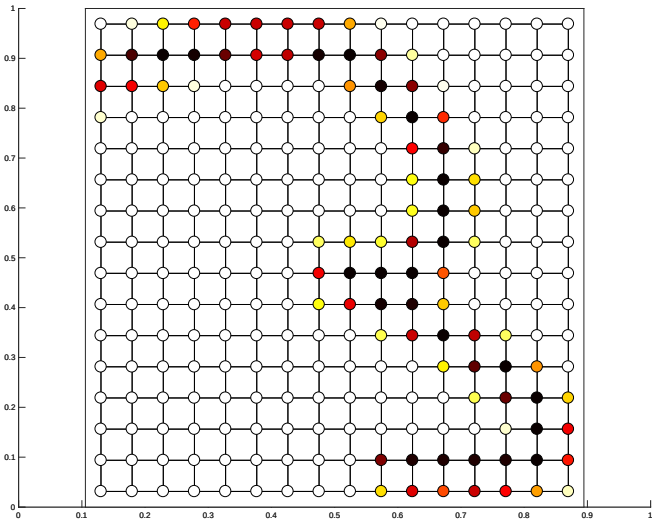

Figure 1.5 Grid graph representing an image from the USPS dataset

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 connections. 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 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
- Scale: total number of nodes is equal to the number of pixels (typically in the order of millions)

While signals defined on the graphs of Boxes 1.1 and 1.2 can already be analyzed using existing signal processing methods, these examples are still important.

Because tools from graph signal processing exactly correspond to those developed in classical signal processing (for some particular graphs), we can view GSP tools as a "natural" generalization of existing methods. As an example, the line graph of Figure 1.3 leads to a well known transform, the size 8 DCT, used for example in the JPEG image compression standard. But slight changes to those weights lead to different transformations (see Chapter 10). As graphs deviate

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

Table 1.1 Classes of problems of interest

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.

1.3 Practical scenarios for GSP

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 in several categories, as shown in Table 1.1,which we discuss next.

1.3.1 Physical networks

Physical networks are systems were nodes correspond to physical devices or components, while links between these nodes are a function of distance between nodes or represent physical communication between them. In a transportation network, nodes may represent hubs or intersections, while links could represent roads or train tracks. In an electric network nodes may include power generators and homes, while link would represent transmission lines. In a communication network such as the Internet, each node may be a computer or other connected device, while each link would represent a communication link.

As seen in the transportation road network shown in Fig. 1.6, one key feature of these physical networks is that the position of the nodes can be mapped to an actual position. In some cases, e.g., sensor networks, there are no obvious

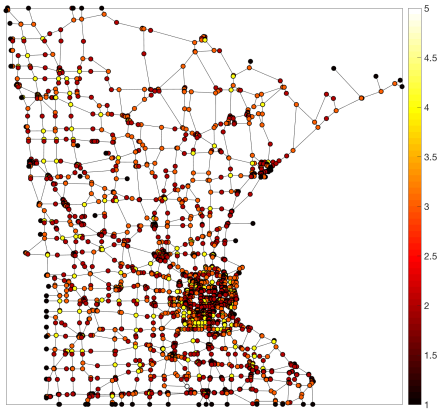

Figure 1.6 Minnesota road network graph

links between 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 use the edge weights that are based on distance between nodes when following the network (i.e., the distance along the road, rather than the distance between nodes). Finally, in some networks, e.g., the Internet, the exact position of the nodes may be known but may not be key in defining a graph representation. Instead, capacity of communication links may be more important.

Box 1.3 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
- Scale: dozens to thousands of nodes

As a concrete example of a physical system, consider **sensor networks**, see Box 1.3. A variety of systems can be described as sensor networks. Examples include sets of distributed temperature sensors (indoors or outdoors), surveillance cameras or even devices carried by a number of users, such as cellphones.

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. 1.7

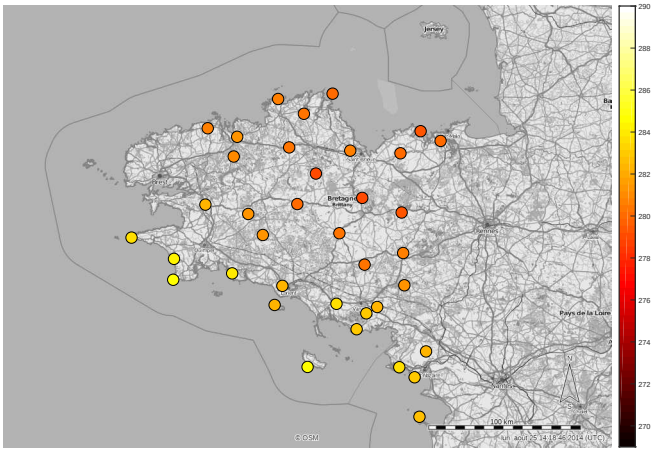

Figure 1.7 Sensor network

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, store and transmit information has continued to 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.

1.3.2 Information networks

Organization of information as a network or graph has existed since at least the first encyclopedia, with its entries and cross-references. The Internet simply made this way of organizing information more explicit (and easier to browse). Indeed, one of the most popular sources of information is wikipedia, itself an online version of an encyclopedia. In typical information networks each node represent a 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 vertex, and the graph is directed, since a 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 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 at the core of Google, where the essential

Box 1.4 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)
- Scale: the number of webpages is estimated to be several billion

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.

1.3.3 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 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
- Scale: The number of nodes corresponds to the size of training set which could be in the order of millions points for some modern datasets

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.

1.3.4 Analyzing complex systems

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

1.3.5 Social networks

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 users and their 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 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.

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 to each user, e.g., geographical location.

1.4 Mathematical models of graphs

In this book we mostly consider scenarios where a graph is given and we develop tools to analyze the graph signals on the given graph. 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.

In the context of the broad field of *Network Science* there has been a significant amount of work to develop probabilistic models of classes of graphs, as tools to derive estimates for various graph properties that are valid for those specific graph classes. In particular these models are used in order to develop closed form expressions for node degree distributions (probability that a node has a certain number of neighbors).

These random models are given a number of N of nodes and describe the

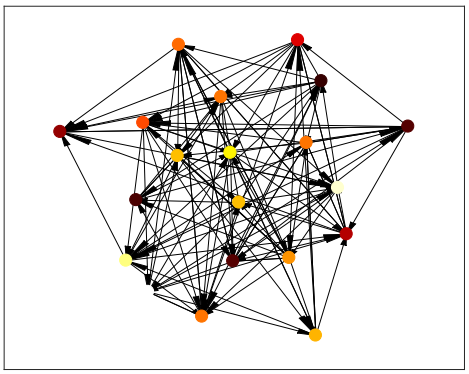

Figure 1.8 Erdos-Renyi Graph

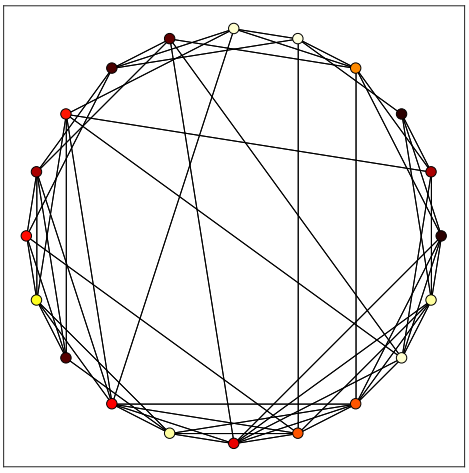

Figure 1.9 Watts-Strogatz or Small World graph

probability that two nodes are connected. We briefly some of the most popular among these models, in order to illustrate the basic concepts and link the specific models to some of the concrete examples of graphs considered in the previous section.

Erdos-Renyi graphs (see Figure 1.4) are generated selecting a probability that two nodes are connected, and applying this to every pair of nodes. While this mathematical models allows results about connectedness and degree distribution to be derived, these types of models do not represent well the structure present in many real-world networks, in the sense that all nodes exhibit the same behavior.

An alternative model that addresses some of the limitations of Erdos-Renyi graphs is the Watts-Strogatz or Small World graph (see Figure 1.9). These models start with a regular graph. First a cycle graph with N nodes such as the one in

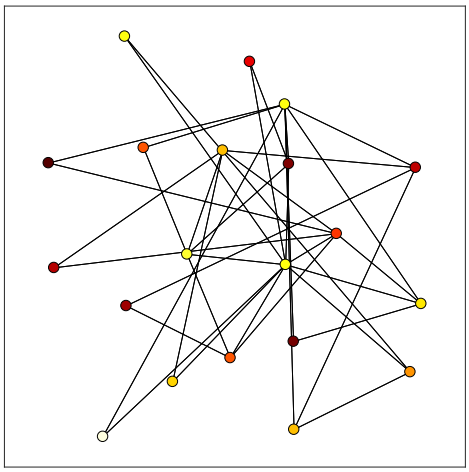

Figure 1.10 Barabasi Albert or Scale-free graph

Figure 1.4 is created. Then connections are added while preserving regularity. In the initial model of Figure 1.4, each node has exactly two neighbors, then additional links are included so that each node connects to its k closest neighbors in the original cycle. Finally with some low probability connections are added between any two nodes. These additional added nodes provide the small world property, where it is possible to find relatively short paths between any two nodes.

Barabasi Albert or Scale-free graphs (see Figure 1.10) 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 probabilty some nodes can have very high degree). Starting with 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.

1.5 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
- Sampling