6.047/6.878 Lecture 18
Regulatory Networks:
Inference, Analysis, Application

Guest Lecture by
Sushmita Roy (2010) / Soheil Feizi (2012)
Scribed by Ben Holmes (2010) / Hamza Fawzi and Sara Brockmueller (2012)

October 26, 2012

Contents

List of Figures

1 Introduction

Living systems are composed of multiple layers that encode information about the system. The primary layers are:

1. Epigenome: Defined by chromatin configuration. The structure of chromatin is based on the way that histones organize DNA. DNA is divided into nucleosome and nucleosome-free regions, forming its final shape and influencing gene expression. [1]
2. Genome: Includes coding and non-coding DNA. Genes defined by coding DNA are used to build RNA, and Cis-regulatory elements regulate the expression of these genes.
3. Transcriptome RNAs (ex. mRNA, miRNA, ncRNA, piRNA) are transcribed from DNA. They have regulatory functions and manufacture proteins.
4. Proteome Composed of proteins. This includes transcription factors, signaling proteins, and metabolic enzymes.

Interactions between these components are all different, but understanding them can put particular parts of the system into the context of the whole. To discover relationships and interactions within and between layers, we can use networks.

1.1 Introducing Biological Networks

Biological networks are composed as follows:

Regulatory Net – set of regulatory interactions in an organism.

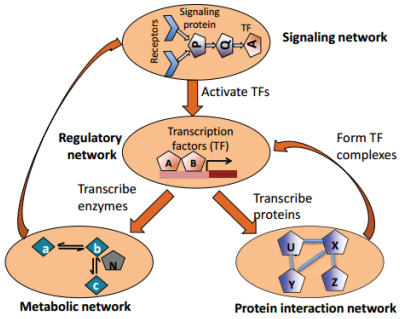
- Nodes are regulators (ex. transcription factors) and associated targets.
- Edges correspond to regulatory interaction, directed from the regulatory factor to its target. They are signed according to the positive or negative effect and weighted according to the strength of the reaction.

Metabolic Net – connects metabolic processes. There is some flexibility in the representation, but an example is a graph displaying shared metabolic products between enzymes.

- Nodes are enzymes.
- Edges correspond to regulatory reactions, and are weighted according to the strength of the reaction.

Signaling Net – represents paths of biological signals.

[1] More in the epigenetics lecture.



(a) Interactions between biological networks.

- Nodes are proteins called signaling receptors.
- Edges are transmitted and received biological signals, directed from transmitter to receiver.

Protein Net – displays physical interactions between proteins.

- Nodes are individual proteins.
- Edges are physical interactions between proteins.

Co-Expression Net – describes co-expression functions between genes. Quite general; represents functional rather than physical interaction networks, unlike the other types of nets. Powerful tool in computational analysis of biological data.

- Nodes are individual genes.
- Edges are co-expression relationships.

Today, we will focus exclusively on regulatory networks. Regulatory networks control context-specific gene expression, and thus have a great deal of control over development. They are worth studying because they are prone to malfunction and causing disease.

1.2 Interactions Between Biological Networks

Individual biological networks (that is, layers) can themselves be considered nodes in a larger network representing the entire biological system. We can, for example, have a signaling network sensing the environment governing the expression of transcription factors. In this example, the network would display that TFs govern the expression of proteins, proteins can play roles as enzymes in metabolic pathways, and so on.

The general paths of information exchange between these networks are shown in figure 2.

1.3 Studying Regulatory Networks

In general, networks are used to represent dependencies among variables. Structural dependencies can be represented by the presence of an edge between nodes - as such, unconnected nodes are conditionally independent. Probabilistically, edges can be assigned a "weight" that represents the strength or the likelihood of the interaction. Networks can also be viewed as matrices, allowing mathematical operations. These frameworks provides an effective way to represent and study biological systems.

These networks are particularly interesting to study because malfunctions can have a large effect. Many diseases are caused by rewirings of regulatory networks. They control context specific expression in development. Because of this, they can be used in systems biology to predict development, cell state, system state, and more. In addition, they encapsulate much of the evolutionary difference between organisms that are genetically similar.

To describe regulatory networks, there are several challenging questions to answer.

Element Identification What are the elements of a network? Elements constituting regulatory networks were identified last lecture. These include upstream motifs and their associated factors.

Network Structure Analysis How are the elements of a network connected? Given a network, structure analysis consists of examination and characterization of important properties. It can be done biological networks but is not restricted to them.

Network Inference How do regulators interact and turn on genes? This is the task of identifying gene edges and characterizing their actions.

Network Applications What can we do with networks once we have them? Applications include predicting function of regulating genes and predicting expression levels of regulated genes.

2 Structure Inference

2.1 Key Questions in Structure Inteference

How to choose network models? A number of models exist for representing networks, a key problem is choosing between them based on data and predicted dynamics.

How to choose learning methods? Two broad methods exist for learning networks. Unsupervised methods attempt to infer relationships for unalabeled datapoints and will be described in sections to come. Supervised methods take a subset of network edges known to be regulatory, and learn a classifier to predict new ones.[2]

How to incorporate data? A variety of data sources can be used to learn and build networks including Motifs, ChIP binding assays, and expression. Data sources are always expanding; expanding availability of data is at the heart of the current revolution in analyzing biological networks.

2.2 Abstract Mathematical Representations for Networks

Think of a network as a function, a black box. Regulatory networks for example, take input expressions of regulators and spit out output expression of targets. Models differ in choosing the nature of functions and assigning meaning to nodes and edges.

Boolean Network This model discretizes node expression levels and interactions. Functions represented by edges are logic gates.

Differential Equation Model These models capture network dynamics. Expression rate changes are function of expression levels and rates of change of regulators. For these it can be very difficult to estimate parameters. Where do you find data for systems out o equilibrium?

Probabilistic Graphical Model These systems model networks as a joint probability distribution over random variables. Edges represent conditional dependencies. Probabilistic graphical models (PGMs) are focused on in the lecture.

2.2.1 Probabilistic Graphical Models

Probabilistic graphical models (PGMs) are trainable and able to deal with noise and thus they are good tools for working with biological data.[3] In PGMs, nodes can be transcription factors or genes and they are modeled by random variables. If you know the joint distribution over these random variables, you can build the network as a PGMs. Since this graph structure is a compact representation of the network, we can work with it easily and accomplish learning tasks. Examples of PGMS include:

[2]Supervised methods will not be addressed today.
[3]These are Dr. Roys models of choice for dealing with biological nets.

Bayesian Network Directed graphical technique. Every node is either a parent or a child. Parents fully determine the state of children but their states may not be available to the experimenter. The network structure describes the full joint probablility distribution of the network as a product of individual distributions for the nodes. By breaking up the network into local potentials, computational complexity is drastically reduced.

Dynamic Bayesian Network Directed graphical technique. Static bayesian networks do not allow cyclic dependencies but we can try to model them with bayesian networks allowing arbitrary dependencies between nodes at different time points. Thus cyclic dependencies are allowed as the network progresses through time and the network joint probability itself can be described as a joint over all times.

Markov Random Field Undirected graphical technique. Models potentials in terms of cliques. Allows modelling of general graphs including cyclic ones with higher order than pairwise dependencies.

Factor Graph Undirected graphical technique. Factor graphs introduce "factor" nodes specifying interaction potentials along edges. Factor nodes can also be introduced to model higher order potentials than pairwise.

It is easiest to learn networks for Bayesian models. Markov random fields and factor graphs require determination of a tricky partition function. To encode network structure, it is only necessary to assign random variables to TFs and genes and then model the joint probability distribution.

Bayesian networks provide compact representations of JPD

The main strength of Bayesian networks comes from the simplicity of their decomposition into parents and children. Because the networks are directed, the full joint probability distribution decomposes into a product of conditional distributions, one for each node in the network.[4]

2.2.2 Network Inference from Expression Data

Using expression data and prior knowledge, the goal of network inference is to produce a network graph. Graphs will be undirected or directed. Regulatory networks for example will often be directed whil expression nets for example will be undirected.

3 Overview of the PGM Learning Task

We have to learn parameters from the data we have. Once we have a set of parameters, we have to use parametrizations to learn structure. Wewill focus on score based approaches to network building, defining a score to be optimized as a metric for network construction.

3.1 Parameter Learning for Bayesian Networks

Maximum Likelihood Chooses parameters to maximize the likelihood of the available data given the model.

In maximum likelihood, compute data likelihood as scores of each ran- dom variable given parents and note that scores can be optimized in- dependently. Depending on the choice of a model, scores will be max- imized in different manners. For gaussian distriubution it is possible to simply compute parameters optimizing score. For more complicated model choices it may be necessary to do gradient descent.

[4] Bayesian networks are parametrized by θ according to our specific choice of network model. With different choices of random variables, we will have different options for parametrizations,θ and therefore different learning tasks:

Discrete Random variables suggest simple θ corresponding to parameter choices for a multinomial distribution.

Continuous Random variables may be modelled with θ corresponding to means and covariances of gaussians or other continuous distribution.

Bayesian Parameter Estimation Treats θ itself as a random variable and chooses the parameters maximizing the posterior probability. These methods require a fixed structure and seek to choose internal parameters maximizing score.

3.1.1 Structure Learning

We can compute best guess parametrizations of structured networks. How do we find structures themselves?

Structure learning proceeds by comparing likelihood of ML parametrizations across different graph structures and in order to seek those structures realizing optimal of ML score.

A Bayesian framework can incorporate prior probabilities over graph structures if given some reason to believe a-priori that some structures are more likely than others.

To perform search in structure learning, we will inevitably have to use a greedy approach because the space of structures is too large to enumerate. Such methods will proceed by an incremental search analogous to gradient descent optimization to find ML parametrizations.

A set of graphs are considered and evaluated according to ML score. Since local optima can exist, it is good to seed graph searches from multiple starting points.

Besides being unable to capture cyclic dependencies as mentioned above, Bayesian networks have certain other limitations.

Indirect Links Since Bayesian networks simply look at statistical dependencies between nodes, it is easy for them to be tricked into putting edges where only indirect relations are in fact present.

Neglected Interactions Especially when structural scores are locally optimized, it is possible that significant biological interactions will be missed entirely. Coexpressed genes may not share proper regulators.

Slow Speed Bayesian methods so far discussed are too slow to work effectively whole-genome data.

3.1.2 Excluding Indirect Links

How to eliminate indirect links? Information theoretic approaches can be used to remove extraneous links by pruning network structures to remove redundant information. Two methods are described.

ARACNE For every triplet of edges, a mutual information score is computed and the ARACNE algorithm excludes edges with the least information subject to certain thresholds above which minimal edges are kept.

MRNET Maximizes dependence between regulators and targets while minimizing the amount of redundant information shared between regulators by stripping edges corresponding to regulators with low variance.

Alternately it is possible to simply look at regulatory motifs and eliminate regulation edges not predicted by common motifs.

3.2 Learning Regulatory Programs for Modules

How to fix omissions for coregulated genes? By learning parameters for regulatory models instead of individual genes, it is possible to exploit the tendency of coexpressed genes to be regulated similarly. Similar to the method of using regulatory motifs to prune redundant edges, by modeling modules at once, we reduce network edge counts while increasing data volume to work with.

With extensions, it is possible to model cyclic dependencies as well. Module networks allow clustering revisitation where genes are reassigned to clusters based on how well hey are predicted by a regulatory program for a module.

Modules however cannot accomodate genes sharing module membership. divide and conquer for speeding up learning

How to speed up learning? Dr. Roy has developed a method to break the large learning problem into smaller tasks using a divide and conquer technique for undirected graphs. By starting with clusters it is possible to infer regulatory networks for individual clusters then cross edges, reassign genes, and iterate.

3.3 Conclusions in Network Inference

Regulatory networks are important but hard to construct in general. By exploiting modularity, it is often possible to find reliable structures for graphs and subgraphs.[5]

Many extensions are on the horizon for regulatory networks. These include inferring causal edges from expression correlations, learning how to share genes between clusters, and others.

4 Applications of Networks

Using linear regression and regression trees, we will try to predict expression from networks. Using collective classification and relaxation labeling, we will try to assign function to unknown network elements.

We would like to use networks to:

1. predict the expression of genes from regulators.

   In expression prediction, the goal is to parametrize a relationship giving gene expression levels from regulator expression levels. It can be solved in various manners including regression and is related to the problem of finding functional networks.

2. predict functions for unknown genes.

4.1 Overview of Functional Models

One model for prediction is a conditional gaussian: a simple model trained by linear regression. A more complex prediction model is a regression tree trained by nonlinear regression.

4.1.1 Conditional Gaussian Models

Conditional gaussian models predict over a continuous space and are trained by a simple linear regression to maximize likelihood of data. They predict targets whose expression levels are means of gaussians over regulators.

Conditional gaussian learning takes a structured, directed net with targets and regulating transcription factors. You can estimate gaussian parameters,μ, σ from the the data by finding parameters maximizing likelihood - after a derivative, the ML approach reduces to solving a linear equation.

From a functional regulatory network derived from multiple data sources [6],Dr, Roy trained a gaussian model for prediction using time course expression data and tested it on a hold-out testing set. In comparisons to predictions by a modle trained from a random network, found out that the network predicted substantially better than random.

The linear model used makes a strong assumption on linearity of interaction. This is probably not a very accurate assumption to make but it appears to work to some extent with the dataset tested.

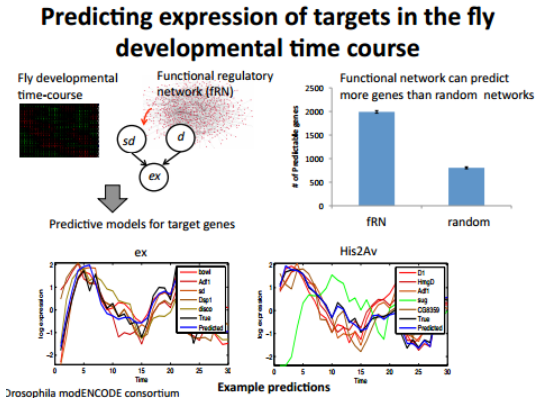
4.1.2 Regression Tree Models

Regression tree models allow the modeler to use a multimodal distribtion incorporating nonlinear dependencies between regulator and target gene expression. The final structure of a regression tree describes expression grammar in terms of a series of choices made at regression tree nodes. Because targets can share regulatory programs, notions of recurring motifs may be incorporated. Regression trees are rich models but tricky to learn. regression trees in predicting expression

In practice, prediction works its way down a regression tree given regulator expression levels. Upon reaching the leaf nodes of the regression tree, a prediction for gene expression is made.

[5] Dr. Roy notes that many algorithms are available for running module network inference with various distributions. Neural net pacakges and Bayesian packages among others are available.

[6] data sources included chromatin, physical binding, expression, motif



(b) Fly development.

4.2 Functional Prediction for Unannotated Nodes

Given a network with an incomplete set of labels, the goal of function annotation is to predict labels for unknown genes. We will use methods falling under the broad category of guilt by association. If we know nothing about a node but that its neighbors are involved in a function, assign that function to the unknown node.

Association can include any notion of network relatedness discussed above such as co-expression, protein-protein interactions and co-regulation. Many methods work, two will be discussed: collective classification and relaxation classification; both of which work for regulatory networks encoded as undirected graphs.

4.2.1 Collective Classification

View functional prediction as a classification problem: Given a node, what is its regulatory class?.

In order to use the graph structure in the prediction problem, we capture properties of the neighborhood of a gene in relational attribute. Since all points are connected in a network, data points are no longer independently distributed - the prediction problem becomes substantially harder than a standard classification problem.

Iterative classification is a simple method with which to solve the classification problem. Starting with an initial guess for unlabeled genes it infers labels iteratively, allowing changed labels to influence node label predictions in a manner similar to gibbs sampling[7]

Relaxation labeling is another approach originally developed to trac terrorist networks. The model uses a suspicion score where nodes are labeled with a suspiciousness according to the suspiciousness of its neighbors. The method is called relaxation labeling because it gradually settles on to a solution according to a learning parameter. It is another instance of iterative learning where genes are assigned probabilities of having a given function.

4.2.2 Regulatory Networks for Function Prediction

For pairs of nodes, compute a regulatory similarity – the interaction quantity – equal to the size of the intersection of their regulators divided by the size of their union. Having this interaction similarity in the form of an undirected graph over netowrk targets, can use clusters derived from a network in final functional classification.

The model is successful in predicting invaginal disk and neural system development. The blue line in Fig. 1b shows the score of every gene predicting its participation in neural system development.

Co-expression an co-regulation can be used side by side to augment the set of genes known to particiapte in neural system development.

[7] see the previous lecture by Manolis describing motif discovery

5 Structural Properties of Networks

Much of the early work on networks was done by scientists outside of biology. Physicists looked at internet and social networks and described their properties. Biologists observed that the same properties were also present in biological networks and the field of biological networks was born. In this section we look at some of these structural properties shared by the different biological networks, as well as the networks that arise in other disciplines as well.

5.1 Degree distribution

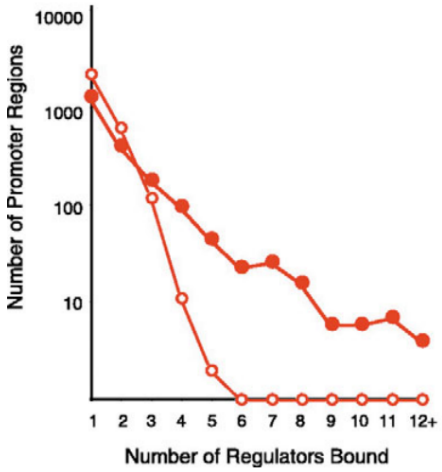


Figure 1: The solid symbols give the in-degree distribution of genes in the regulatory network of *S. cerevisiae* (the in-degree of a gene is the number of transcription factors that bind to the promoter of this gene). The open symbols give the in-degree distribution in the comparable random network. Figure taken from [4].

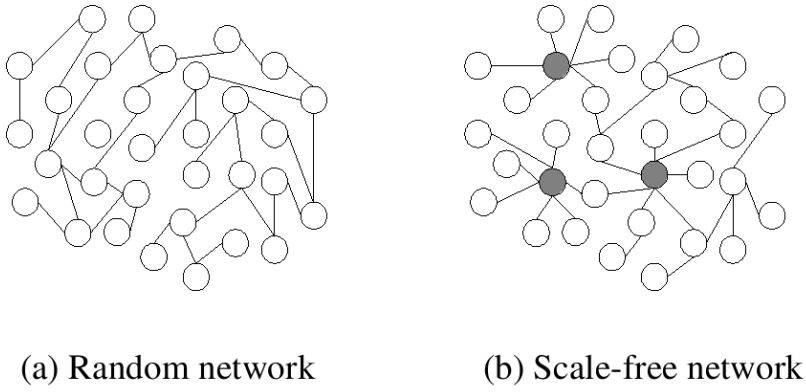
In a network, the *degree* of a node is the number of neighbors it has, i.e., the number of nodes it is connected to by an edge. The *degree distribution* of the network gives the number of nodes having degree d for each possible value of $d = 1, 2, 3, \ldots$. For example figure 1 gives the degree distribution of the *S. cerevisiae* gene regulatory network. It was observed that the degree distribution of biological networks follow a power law, i.e., the number of nodes in the network having degree d is approximately $cd^{-\gamma}$ where c is a normalization constant and γ is a positive coefficient. In such networks, most nodes have a small number of connections, except for a few nodes which have very high connectivity.

This property –of power law degree distribution– was actually observed in many different networks across different disciplines (e.g., social networks, the World Wide Web, etc.) and indicates that those networks are not "random": indeed random networks (constructed from the Erdős-Renyi model) have a degree distribution that follows a Poisson distribution where almost all nodes have approximately the same degree and nodes with higher or smaller degree are very rare [6] (see figure 2).

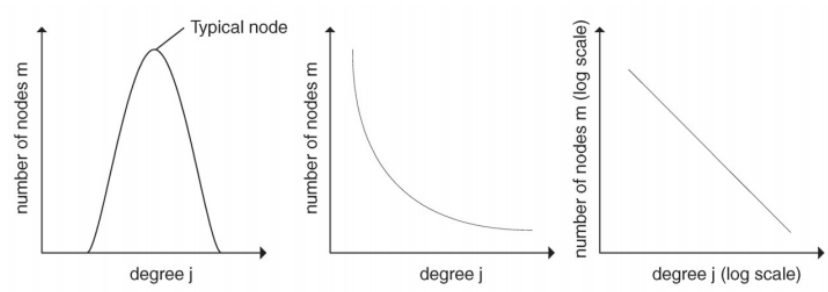
Networks that follow a power law degree distribution are known as **scale-free networks**. The few nodes in a scale-free network that have very large degree are called *hubs* and have very important interpretations. For example in gene regulatory networks, hubs represent transcription factors that regulate a very large number of genes. Scale-free networks have the property of being highly resilient to failures of "random" nodes, however they are very vulnerable to coordinated failures (i.e., the network fails if one of the hub nodes fails, see [1] for more information).

In a regulatory network, one can identify four levels of nodes:

1. Influential, master regulating nodes on top. These are hubs that each indirectly control many targets.
2. Bottleneck regulators. Nodes in the middle are important because they have a maximal number of direct targets.



(a) Scale-free graph vs. a random graph (figure taken from [10]).



(b) Degree distribution of scale-free network vs. random network (figure taken from [3]).

Figure 2: Scale-free vs. random Erdős-Renyi networks

3. Regulators at the bottom tend to have fewer targets but nonetheless they are often biologically essential!
4. Targets.

5.2 Network motifs

Network motifs are subgraphs of the network that occur significantly more than random. Some will have interesting functional properties and are presumably of biological interest.

Figure 3 shows regulatory motifs from the yeast regulatory network. Feedback loops allow control of regulator levels and feedforward loops allow acceleration of response times among other things.

6 Network clustering

An important problem in network analysis is to be able to **cluster** or **modularize** the network in order to identify subgraphs that are densely connected (see e.g., figure 4a). In the context of gene interaction networks, these clusters could correspond to genes that are involved in similar functions and that are co-regulated.

There are several known algorithms to achieve this task. These algorithms are usually called *graph partitioning algorithms* since they partition the graph into separate modules. Some of the well-known algorithms include:

- Markov clustering algorithm [5]: The Markov Clustering Algorithm (MCL) works by doing a random walk in the graph and looking at the steady-state distribution of this walk. This steady-state distribution allows to cluster the graph into densely connected subgraphs.

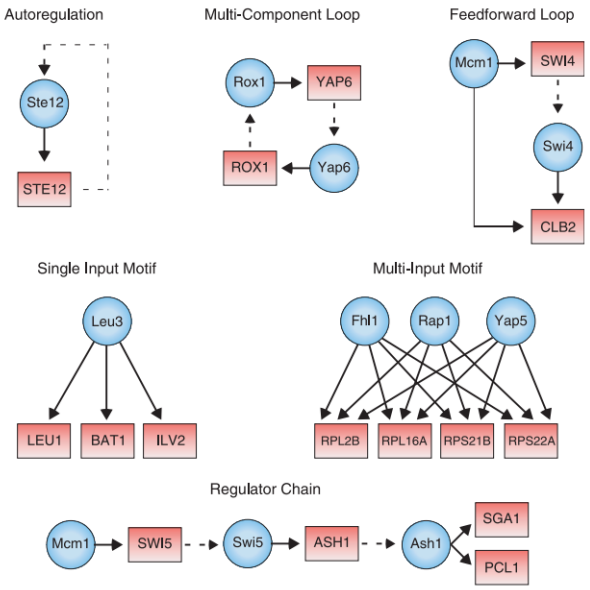


Figure 3: Network motifs in regulatory networks: Feed-forward loops involved in speeding-up response of target gene. Regulators are represented by blue circles and gene promoters are represented by red rectangles (figure taken from [4])

- Girvan-Newman algorithm [2]: The Girvan-Newman algorithm uses the number of shortest paths going through a node to compute the *essentiality* of an edge which can then be used to cluster the network.
- Spectral partitioning algorithm

In this section we will look in detail at the spectral partitioning algorithm. We refer the reader to the references [2, 5] for a description of the other algorithms.

The spectral partitioning algorithm relies on a certain way of representing a network using a matrix. Before presenting the algorithm we will thus review how to represent a network using a matrix, and how to extract information about the network using matrix operations.

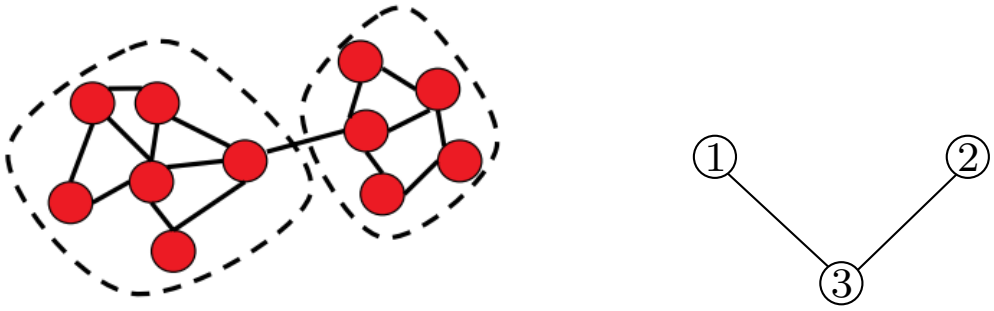
6.1 An algebraic view to networks

Adjacency matrix One way to represent a network is using the so-called *adjacency matrix*. The adjacency matrix of a network with n nodes is an $n \times n$ matrix A where $A_{i,j}$ is equal to one if there is an edge between nodes i and j, and 0 otherwise. For example, the adjacency matrix of the graph represented in figure 4b is given by:

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \tag{1}$$

If the network is weighted (i.e., if the edges of the network each have an associated weight), the definition of the adjacency matrix is modified so that $A_{i,j}$ holds the weight of the edge between i and j if the edge exists, and zero otherwise.

Laplacian matrix For the clustering algorithm that we will present later in this section, we will need to count the number of edges between the two different groups in a partitioning of the network. For example, in Figure 4a, the number of edges between the two groups is 1. The *Laplacian matrix* which we will introduce now comes in handy to represent this quantity algebraically. The Laplacian matrix L of a network on n nodes is a $n \times n$ matrix L that is very similar to the adjacency matrix A except for sign changes and for the diagonal elements. Whereas the diagonal elements of the adjacency matrix are always equal to zero (since we



(a) A partition of a network into two groups.

(b) A simple network on 3 nodes. The adjacency matrix of this graph is given in equation (1).

Figure 4

do not have self-loops), the diagonal elements of the Laplacian matrix hold the *degree* of each node (where the degree of a node is defined as the number of edges incident to it). Also the off-diagonal elements of the Laplacian matrix are set to be -1 in the presence of an edge, and zero otherwise. In other words, we have:

$$L_{i,j} = \begin{cases} \text{degree}(i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and there is an edge between } i \text{ and } j \\ 0 & \text{if } i \neq j \text{ and there is no edge between } i \text{ and } j \end{cases} \tag{2}$$

For example the Laplacian matrix of the graph of figure 4b is given by (we emphasized the diagonal elements in bold):

$$L = \begin{bmatrix} \mathbf{1} & 0 & -1 \\ 0 & \mathbf{1} & -1 \\ -1 & -1 & \mathbf{2} \end{bmatrix}$$

Some properties of the Laplacian matrix The Laplacian matrix of any network enjoys some nice properties that will be important later when we look at the clustering algorithm. We briefly review these here.

The Laplacian matrix L is always **symmetric**, i.e., $L_{i,j} = L_{j,i}$ for any i, j. An important consequence of this observation is that all the eigenvalues of L are real (i.e., they have no complex imaginary part). In fact one can even show that the eigenvalues of L are all nonnegative[8] The final property that we mention about L is that all the rows and columns of L sum to zero (this is easy to verify using the definition of L). This means that the smallest eigenvalue of L is always equal to zero, and the corresponding eigenvector is $s = (1, 1, \ldots, 1)$.

Counting the number of edges between groups using the Laplacian matrix Using the Laplacian matrix we can now easily count the number of edges that separate two disjoint parts of the graph using simple matrix operations. Indeed, assume that we partitioned our graph into two groups, and that we define a vector s of size n which tells us which group each node i belongs to:

$$s_i = \begin{cases} 1 & \text{if node } i \text{ is in group 1} \\ -1 & \text{if node } i \text{ is in group 2} \end{cases}$$

Then one can easily show that the total number of edges between group 1 and group 2 is given by the quantity $\frac{1}{4} s^T L s$ where L is the Laplacian of the network.

[8] One way of seeing this is to notice that L is diagonally dominant and the diagonal elements are strictly positive (for more details the reader can look up "diagonally dominant" and "Gershgorin circle theorem" on the Internet).

To see why this is case, let us first compute the matrix-vector product Ls. In particular let us fix a node i say in group 1 (i.e., $s_i = +1$) and let us look at the i'th component of the matrix-vector product Ls. By definition of the matrix-vector product we have:

$$(Ls)_i = \sum_{j=1}^{n} L_{i,j} s_j.$$

We can decompose this sum into three summands as follows:

$$(Ls)_i = \sum_{j=1}^{n} L_{i,j} s_j = L_{i,i} s_i + \sum_{j \text{ in group 1}} L_{i,j} s_j + \sum_{j \text{ in group 2}} L_{i,j} s_j$$

Using the definition of the Laplacian matrix we easily see that the first term corresponds to the degree of i, i.e., the number of edges incident to i; the second term is equal to the negative of the number of edges connecting i to some other node in group 1, and the third term is equal to the number of edges connecting i to some node ingroup 2. Hence we have:

$$(Ls)_i = \text{degree}(i) - (\#\text{ edges from } i \text{ to group 1}) + (\#\text{ edges from } i \text{ to group 2})$$

Now since any edge from i must either go to group 1 or to group 2 we have

$$\text{degree}(i) = (\#\text{ edges from } i \text{ to group 1}) + (\#\text{ edges from } i \text{ to group 2}).$$

Thus combining the two equations above we get:

$$(Ls)_i = 2 \times (\#\text{ edges from } i \text{ to group 2}).$$

Now to get the total number of edges between group 1 and group 2, we simply sum the quantity above over all nodes i in group 1:

$$(\#\text{ edges between group 1 and group 2}) = \frac{1}{2} \sum_{i \text{ in group 1}} (Ls)_i$$

We can also look at nodes in group 2 to compute the same quantity and we have:

$$(\#\text{ edges between group 1 and group 2}) = -\frac{1}{2} \sum_{i \text{ in group 2}} (Ls)_i$$

Now averaging the two equations above we get the desired result:

$$\begin{aligned}(\#\text{ edges between group 1 and group 2}) &= \frac{1}{4} \sum_{i \text{ in group 1}} (Ls)_i - \frac{1}{4} \sum_{i \text{ in group 2}} (Ls)_i \\ &= \frac{1}{4} \sum_i s_i (Ls)_i \\ &= \frac{1}{4} s^T Ls\end{aligned}$$

where s^T is the row vector obtained by transposing the column vector s.

6.2 The spectral clustering algorithm

We will now see how the linear algebra view of networks given in the previous section can be used to produce a "good" partitioning of the graph. In any good partitioning of a graph the number of edges between the two groups must be relatively small compared to the number of edges within each group. Thus one way of addressing the problem is to look for a partition so that the number of edges between the two groups is minimal. Using the tools introduced in the previous section, this problem is thus equivalent to finding a

vector $s \in \{-1, +1\}^n$ taking only values -1 or $+1$ such that $\frac{1}{4}s^T Ls$ is minimal, where L is the Laplacian matrix of the graph. In other words, we want to solve the minimization problem:

$$\underset{s\in\{-1,+1\}^n}{\text{minimize}} \quad \frac{1}{4}s^T Ls$$

If s^* is the optimal solution, then the optimal partioning is to assign node i to group 1 if $s_i = +1$ or else to group 2.

This formulation seems to make sense but there is a small glitch unfortunately: the solution to this problem will always end up being $s = (+1, \ldots, +1)$ which corresponds to putting all the nodes of the network in group 1, and no node in group 2! The number of edges between group 1 and group 2 is then simply zero and is indeed minimal!

To obtain a meaningful partition we thus have to consider partitions of the graph that are nontrivial. Recall that the Laplacian matrix L is always symmetric, and thus it admits an eigendecomposition:

$$L = U\Sigma U^T = \sum_{i=1}^{n} \lambda_i u_i u_i^T$$

where Σ is a diagonal matrix holding the nonnegative eigenvalues $\lambda_1, \ldots, \lambda_n$ of L and U is the matrix of eigenvectors and it satisfies $U^T = U^{-1}$.

The cost of a partitioning $s \in \{-1, +1\}^n$ is given by

$$\frac{1}{4}s^T Ls = \frac{1}{4}s^T U\Sigma U^T s = \frac{1}{4}\sum_{i=1}^{n} \lambda_i \alpha_i^2$$

where $\alpha = U^T s$ give the decomposition of s as a linear combination of the eigenvectors of L: $s = \sum_{i=1}^{n} \alpha_i u_i$.

Recall also that $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. Thus one way to make the quantity above as small as possible (without picking the trivial partitioning) is to concentrate all the weight on λ_2 which is the smallest nonzero eigenvalue of L. To achieve this we simply pick s so that $\alpha_2 = 1$ and $\alpha_k = 0$ for all $k \neq 2$. In other words, this corresponds to taking s to be equal to u_2 the second eigenvector of L. Since in general the eigenvector u_2 is not integer-valued (i.e., the components of u_2 can be different than -1 or $+1$), we have to convert first the vector u_2 into a vector of $+1$'s or -1's. A simple way of doing this is just to look at the signs of the components of u_2 instead of the values themselves. Our partition is thus given by:

$$s = \text{sign}(u_2) = \begin{cases} 1 & \text{if } (u_2)_i \geq 0 \\ -1 & \text{if } (u_2)_i < 0 \end{cases}$$

To recap, the spectral clustering algorithm works as follows:

Spectral partitioning algorithm

- Input: a network
- Output: a partitioning of the network where each node is assigned either to group 1 or group 2 so that the number of edges between the two groups is small

1. Compute the Laplacian matrix L of the graph given by:

$$L_{i,j} = \begin{cases} \text{degree}(i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and there is an edge between } i \text{ and } j \\ 0 & \text{if } i \neq j \text{ and there is no edge between } i \text{ and } j \end{cases}$$

2. Compute the eigenvector u_2 for the second smallest eigenvalue of L.
3. Output the following partition: Assign node i to group 1 if $(u_2)_i \geq 0$, otherwise assign node i to group 2.

We next give an example where we apply the spectral clustering algorithm to a network with 8 nodes.

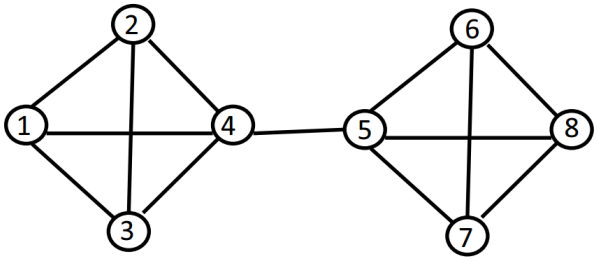


Figure 5: A network with 8 nodes.

Example We illustrate here the partitioning algorithm described above on a simple network of 8 nodes given in figure 5. The adjacency matrix and the Laplacian matrix of this graph are given below:

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \qquad L = \begin{bmatrix} 3 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & 3 \end{bmatrix}$$

Using the `eig` command of Matlab we can compute the eigendecomposition $L = U\Sigma U^T$ of the Laplacian matrix and we obtain:

$$U = \begin{bmatrix} 0.3536 & \mathbf{-0.3825} & 0.2714 & -0.1628 & -0.7783 & 0.0495 & -0.0064 & -0.1426 \\ 0.3536 & \mathbf{-0.3825} & 0.5580 & -0.1628 & 0.6066 & 0.0495 & -0.0064 & -0.1426 \\ 0.3536 & \mathbf{-0.3825} & -0.4495 & 0.6251 & 0.0930 & 0.0495 & -0.3231 & -0.1426 \\ 0.3536 & \mathbf{-0.2470} & -0.3799 & -0.2995 & 0.0786 & -0.1485 & 0.3358 & 0.6626 \\ 0.3536 & \mathbf{0.2470} & -0.3799 & -0.2995 & 0.0786 & -0.1485 & 0.3358 & -0.6626 \\ 0.3536 & \mathbf{0.3825} & 0.3514 & 0.5572 & -0.0727 & -0.3466 & 0.3860 & 0.1426 \\ 0.3536 & \mathbf{0.3825} & 0.0284 & -0.2577 & -0.0059 & -0.3466 & -0.7218 & 0.1426 \\ 0.3536 & \mathbf{0.3825} & 0.0000 & 0.0000 & 0.0000 & 0.8416 & -0.0000 & 0.1426 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{0.3542} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4.0000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4.0000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4.0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4.0000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4.0000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5.6458 \end{bmatrix}$$

We have highlighted in bold the second smallest eigenvalue of L and the associated eigenvector. To cluster the network we look at the sign of the components of this eigenvector. We see that the first 4 components are negative, and the last 4 components are positive. We will thus cluster the nodes 1 to 4 together in the same group, and nodes 5 to 8 in another group. This looks like a good clustering and in fact this is the "natural" clustering that one considers at first sight of the graph.

Did You Know?

The mathematical problem that we formulated as a motivation for the spectral clustering algorithm is to find a partition of the graph into two groups with a minimimal number of edges between the two groups. The spectral partitioning algorithm we presented does not always give an optimal solution to this problem but it usually works well in practice.

Actually it turns out that the problem as we formulated it can be solved exactly using an efficient algorithm. The problem is sometimes called the **minimum cut** problem since we are looking to cut a minimum number of edges from the graph to make it disconnected (the edges we cut are those between group 1 and group 2). The minimum cut problem can be solved in polynomial time in general, and we refer the reader to the Wikipedia entry on *minimum cut* [9] for more information.

The problem however with minimum cut partitions it that they usually lead to partitions of the graph that are not balanced (e.g., one group has only 1 node, and the remaining nodes are all in the other group). In general one would like to impose additional constraints on the clusters (e.g., lower or upper bounds on the size of clusters, etc.) to obtain more realistic clusters. With such constraints, the problem becomes harder, and we refer the reader to the Wikipedia entry on *Graph partitioning* [8] for more details.

FAQ

Q: How to partition the graph into more than two groups?

A: In this section we only looked at the problem of partitioning the graph into two clusters. What if we want to cluster the graph into more than two clusters? There are several possible extensions of the algorithm presented here to handle k clusters instead of just two. The main idea is to look at the k eigenvectors for the k smallest nonzero eigenvalues of the Laplacian, and then to apply the k-means clustering algorithm appropriately. We refer the reader to the tutorial [7] for more information.

References

[1] R. Albert. Scale-free networks in cell biology. *Journal of cell science*, 118(21):4947–4957, 2005.

[2] M. Girvan and M.E.J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[3] O. Hein, M. Schwind, and W. König. Scale-free networks: The impact of fat tailed degree distribution on diffusion and communication processes. *Wirtschaftsinformatik*, 48(4):267–275, 2006.

[4] T.I. Lee, N.J. Rinaldi, F. Robert, D.T. Odom, Z. Bar-Joseph, G.K. Gerber, N.M. Hannett, C.T. Harbison, C.M. Thompson, I. Simon, et al. Transcriptional regulatory networks in saccharomyces cerevisiae. *Science Signalling*, 298(5594):799, 2002.

[5] S.M. van Dongen. *Graph clustering by flow simulation*. PhD thesis, University of Utrecht, The Netherlands, 2000.

[6] M. Vidal, M.E. Cusick, and A.L. Barabasi. Interactome networks and human disease. *Cell*, 144(6):986–998, 2011.

[7] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[8] Wikipedia. Graph partitioning, 2012.

[9] Wikipedia. Minimum cut, 2012.

[10] Wikipedia. Scale-free network, 2012.