Stat260/CS294: Spectral Graph Methods **Lecture 1 - 01/22/2015**

Lecture: Introduction and Overview

Lecturer: Michael Mahoney *Scribe: Michael Mahoney*

Warning: these notes are still very rough. They provide more details on what we discussed in class, but there may still be some errors, incomplete/imprecise statements, etc. in them.

1 Introduction

1.1 Basic motivating background

The course will cover several topics in *spectral graph methods*. By that, I mean that it will cover not spectral graph theory per se, nor will it cover the application of spectral graph methods per se. In addition, spectral methods is a more general topic, and graph methods is a more general topic. Spectral graph theory uses eigenvectors and eigenvalues (and related quantities) of matrices associated with graphs to say things about those graphs. It is a topic which has been studied from a wide range of perspectives, e.g., theoretical computer science, scientific computing, machine learning, statistics, etc., and as such it is a topic which can be viewed from a wide range of approaches.

The reason for the focus on spectral graph methods is that a wide range of problems are obviously spectral graph methods, and thus they are useful in practice as well as interesting in theory; but, in addition, many other methods that are not obviously spectral graph methods really are spectral graph methods under the hood. We'll get to what I mean by that, but for now think of running some procedure that seems to work, and if one were to perform a somewhat rigorous algorithmic or statistical analysis, it would turn out that that method essentially boiled down to a spectral graph method. As an example, consider the problem of viral propagation on a social network, which is usually described in terms of some sort of infectious agent, but which has strong connections with spectral graph methods.

Our goal will be to understand—by drawing strength from each of the wide range of approaches that have been brought to bear on these problems—when/why spectral graph methods are useful in practical machine learning and data analysis applications, and when/why they (or a vanilla variant of them) are not useful. In the latter case, of course, we'll be interested in understanding whether a better understanding of spectral graph methods can lead to the development of improved algorithmic and statistical techniques—both for large-scale data as well as for small-scale data. Relatedly, we will be interested in whether other methods can perform better or whether the data are just "bad" in some sense.

1.2 Types of data and types of problems

Data comes from all sorts of places, and it can be a challenge to find a good way to represent the data in order to obtain some sort of meaningful insight from the data. Two popular ways to model

data are as *matrices* and as *graphs*.

- Matrices often arise when there are n things, each of which is described by m features. In this case, we have an $m \times n$ matrix A, where each column is a data point described by a bunch of features (or vice versa) and where each row is a vector describing the value of that feature at each data point. Alternatively, matrices can arise when there are n things and we have information about the correlations (or other relationships) between them.
- Graphs often arise when there are n things and the pairwise relationships between them are though to be particularly important. Let's specify a graph by $G = (V, E)$, where V is the set of vertices and E is the set of edges, which are pairs of vertices. (Later they can be weighted, etc., but for now let's say they are undirected, unweighted, etc.) Examples of data graphs include the following.
  - Discretizations of partial differential equations and other physical operators give rise to graphs, where the nodes are points in a physical medium and edges correspond to some sort of physical interaction.
  - Social networks and other internet applications give rise to graphs, where the nodes are individuals and there is an edge between two people if they are friends or have some other sort of interaction.
  - Non-social networks give rise to graphs, where, e.g., devised, routers, or computers are nodes and where there is an edge between two nodes if they are connected and/or have traffic between them.
  - Graphs arise more generally in machine learning and data analysis applications. For example, given a bunch of data points, each of which is a feature vector, we could construct a graph, where the nodes correspond to data points and there is an edge between two data points if they are close in some sense (or a soft version of this, which is what rbf kernels do).

In the same way as we can construct graphs from matrices, we can also construct matrices from graphs. We will see several examples below (e.g., adjacency matrices, Laplacians, low-rank embedding matrices, etc.). Spectral graph methods involve using eigenvectors and eigenvalues of matrices associated with graphs to do stuff.

In order to do stuff, one runs some sort of algorithmic or statistical methods, but it is good to keep an eye on the types of problems that might want to be solved. Here are several canonical examples.

- Graph partitioning: finding clusters/communities. Here, the data might be a bunch of data points (put a picture: sprinkled into a left half and a right half) or it might be a graph (put a picture: two things connected by an edge). There are a million ways to do this, but one very popular one boils down to computing an eigenvector of the so-called Laplacian matrix and using that to partition the data. Why does such a method work? One answer, from TCS, is that is works since it is a relaxation of a combinatorial optimization problem for which there are worst-case quality-of-approximation guarantees. Another answer, from statistics and machine learning, is that is can be used to recover hypothesized clusters, say from a stochastic blockmodel (where the graph consists of several random graphs put together) or from a low-dimensional manifold (upon which the data points sit).

- Prediction: e.g., regression and classification. Here, there is a similar picture, and one popular procedure is to run the same algorithm, compute the same vector and use it to classify the data. In this case, one can also ask: why does such a method work? One answer that is commonly given is that if there are meaningful clusters in the data, say drawn from a manifold, then the boundaries between the clusters correspond to low-density regions; or relatedly that class labels are smooth in the graph topology, or some notion of distance in $\mathbb{R}^n$. But, what if the data are from a discrete place? Then, there is the out-of-sample extension question and a bunch of other issues.

- Centrality and ranking. These are two (different but sometimes conflated) notions from sociology and social networks having to do with how "important" or "central" is an individual/node and relatedly how to rank individuals/nodes. One way to do this is to choose the highest degree node, but this is relatively easy to spam and might not be "real" in other ways, and so there are several related things that go by the name of spectral ranking, eigenvector centrality, and so on. The basic idea is that a node is important if important nodes thing it is important. This suggests looking at loops and triangles in a graph, and when this process is iterated you get random walks and diffusions on the graph. It's not obvious that this has very strong connections with the clustering, classification, etc. problems described above, but it does. Basically, you compute the same eigenvector and use it to rank.

- Encouraging or discouraging "viral propagation." Here, one is given, say, a social network, and one is interested in some sort of iterative process, and one wants to understand its properties. Two examples are the following: there might be a virus or other infectious agent that goes from node to node making other agents sick; or there might be some sort of "buzz" about a new movie or new tennis shoes, and this goes from individual to individual. Both of these are sometimes called viral propagation, but there are important differences, not the least of which is that in the former people typically want to stop the spread of the virus, while in the latter people want to encourage the spread of the virus to sell more tennis shoes.

1.3 Examples of graphs

When algorithms are run on data graphs—some of which might be fairly nice but some of which might not, it can be difficult to know why the algorithm performs as it does. For example, would it perform that way on every possible graph? Similarly, if we are not in some asymptotic limit or if worst-case analysis is somewhat too coarse, then what if anything does the method reveal about the graph? To help address these and related questions, it helps to have several examples of graphs in mind and to see how algorithms perform on those graphs. Here are several good examples.

- A discretization of nice low-dimensional space, e.g., the integers/lattice in some fixed dimension: $\mathbb{Z}_d$, $\mathbb{Z}_d^2$, and $\mathbb{Z}_d^3$.

- A star, meaning a central node to which all of the other nodes are attached.

- A binary tree.

- A complete graph or clique.

- A constant-degree expander, which is basically a very sparse graph that has no good partitions. Alternatively, it can be viewed as a sparse version of the complete graph.

- A hypercube on 2^n vertices.
- A graph consisting of two complete graphs or two expanders or two copies of $\mathbb{Z}_d^2$ that are weakly connected by, say, a line graph.
- A lollipop, meaning a complete graph of expander, with a line graph attached, where the line/stem can have different lengths.

Those are common constructions when thinking about graphs. The following are examples of constructions that are more common in certain network applications.

- An Erdos-Renyi random graph, G_{np}, for $p = 3/n$ or $p \gtrsim \log(n)/n$
- A "small world" graph, which is basically a ring plus a 3 regular random graph.
- A heavy-tailed random graph, with or without min degree assumption, or one constructed from a preferential attachment process.

In addition to things that are explicitly graphs, it also helps to have several examples of matrix-based data to have in mind from which graphs can be constructed. These are often constructed from some sort of nearest-neighbor process. Here are several common examples.

- A nice region of a low-dimensional subspace of $\mathbb{R}^n$ or of a nice low-dimensional manifold embedded in $\mathbb{R}^n$.
- A full-dimensional Gaussian in $\mathbb{R}^n$. Here, most of the mass is on the shell, but what does the graph corresponding to this "look like"?
- Two low-dimensional Gaussians in $\mathbb{R}^n$. This looks like a dumbbell, with two complete graphs at the two ends or two copies of $\mathbb{Z}_d^n$ at the ends, depending on how parameters are set.

1.4 Some questions to consider

Here are a few questions to consider.

- If the original data are vectors that form a matrix, how sensitive are these methods to the details of the graph construction? (Answer: in theory, no; in practice, often yes.)
- If the original data are represented by a graph, how sensitive are these methods to a bit of noise in the graph? (Answer: in theory, no; in practice, often yes.)
- How good a guide is worst case cs and asymptotic statistical theory? (Answer: in theory, good; in practice, often not, but it depends on what is the reference state, e.g., manifold versus stochastic blockmodel.)
- What if you are interested in a small part of a very large graph? E.g., you and your 100 closest friends on a social network, as opposed to you and your 10^9 closest friends on that social network. Do you get the same results if you run some sort of local algorithm on a small part of the graph as you do if you run a global algorithm on a subgraph that is cut out? (Typically no, unless you are very careful.)

1.5 Matrices for graphs

Let $G = (V, E, W)$ be an undirected, possibly weighted, graph. There are many matrices that one can associate with a graph. Two of the most basic are the *adjacency matrix* and the *diagonal degree matrix*.

Definition 1 *Given $G = (V, E, W)$, the adjacency matrix $A \in \mathbb{R}^{n \times n}$ is defined to be*

$$A_{ij} = \begin{cases} W_{ij} & \text{if } (ij) \in E \\ 0 & \text{otherwise} \end{cases},$$

and the diagonal degree matrix $D \in \mathbb{R}^{n \times n}$ is defined to be

$$D_{ij} = \begin{cases} \sum_k W_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}.$$

Note that for undirected graphs, i.e., when W_{ij} equals 1 or 0 depending on whether or not there is an edge between nodes i and j, the adjacency matrix specifies which edges are connected and the diagonal degree matrix gives the degree of i^{th} node at the $(ii)^{th}$ diagonal position.

(Given this setup, it shouldn't be surprising that most spectral graph methods generalize in nice ways from unweighted to weighted graphs. Of interest also are things like time-evolving graphs, directed graphs, etc. In those cases, the situation is more subtle/complex. Typically, methods for those more complex graphs boil down to methods for simpler undirected, static graphs.)

Much of what we will discuss has strong connections with spectral graph theory, which is an area that uses eigenvectors and eigenvalues of matrices associated with the graph to understand properties of the graph. To begin, though, we should note that it shouldn't be obvious that eigenstuff should reveal interesting graph properties—after all, graphs by themselves are essentially combinatorial things and most traditional graph problems and algorithms don't mention anything having to do with eigenvectors. In spite of this, we will see that eigenstuff reveals a lot about graphs that are useful in machine learning and data analysis applications, and we will want to understand why this is the case and how we can take advantage of it in interesting ways.

Such an approach of using eigenvectors and eigenvalues is most useful when used to understand a natural operator of natural quadratic form associated with the graphs. Perhaps surprisingly, adjacency matrices and diagonal degree matrices are not so useful in that sense—but they can be used to construct other matrices that are more useful in that sense.

One natural and very useful operator to associate with a graph G is the following.

Definition 2 *Given $G = (V, E, W)$, the* diffusion operator *is*

$$W = D^{-1}A \quad \textit{(or } M = AD^{-1}\textit{, if you multiply from the other side).}$$

This matrix describes the behavior of diffusions and random walks on G. In particular, if $x \in \mathbb{R}^n$ is a row vector that gives the probability that a particle is at each vertex of G, and if the particle then moves to a random neighbor, then pW is the new probability distribution of the particle. If the graph G is regular, meaning that it is degree-homogeneous, then W is a rescaling of A, but otherwise it can be very different. Although we won't go into too much detail right now, note that

applying this operator to a vector can be interpreted as doing one step of a diffusion or random walk process. In this case, one might want to know what happens if we iteratively apply an operator like W. We will bet back to this.

One natural and very useful quadratic form to associate with a graph G is the following.

Definition 3 *Given $G = (V, E, W)$, the* Laplacian matrix *(or combinatorial Laplacian matrix) is*

$$L = D - A.$$

Although we won't go into detail, the Laplacian has an interpretation in terms of derivatives. (This is most common/obvious in continuous applications, where it can be used to measure the smoothness of the Laplacian and/or of some continuous place from which the gradient was constructed—if it was—in a nice way, which is often *not* the case.) Given a function or a vector $x \in \mathbb{R}^n$, the Laplacian quadratic form is

$$x^T Lx = \sum_{(ij)\in E} (x_i - x_j)^2.$$

This is a measure of smoothness of the vector/function x—smoothness of x, in some sense, conditioned on the graph structure. (That is, it is a statement about the graph itself, independent of how it was constructed. This is of interest by itself but also for machine learning and data analysis application, e.g., since labels associated with the nodes that correspond to a classification function might be expected to be smooth.)

Alternatively, we can define the *normalized Laplacian matrix*.

Definition 4 *Given $G = (V, E, W)$, the* normalized Laplacian matrix *is*

$$\mathcal{L} = L = D^{-1/2} L D^{-1/2} = I - D^{-1/2} L D^{-1/2}.$$

(Note that I have already started to be a little sloppy, by using the same letter to mean two different things. I'll point out as we go where this matters.) As we will see, for degree-homogeneous graphs, these two Laplacians are essentially the same, but for degree-heterogeneous graphs, they are quite different. As a general rule, the latter is more appropriate for realistic degree-heterogeneous graphs, but it is worth keeping the two in mind, since there are strong connections between them and how they are computed. Similar smoothness, etc. interpretations hold for the normalized Laplacian, and this is important in many applications.

1.6 An overview of some ideas

Here is a vanilla version of a spectral graph algorithm that will be central to a lot of what we do. We'll be more precise and go into a lot more detail later. The algorithm takes as input a graph, as specified by a Laplacian matrix, L or $\mathcal{L}$.

1. Compute, exactly or approximately, the leading nontrivial eigenvector of L or $\mathcal{L}$.
2. Use that vector to split the nodes of the graph into a left half and a right half.

Those two pieces can be the two clusters, in which case this algorithm is a vanilla version of spectral graph partitioning; or with some labels that can be used to make predictions for classification or regression; or we can rank starting from the left and going to the right; or we can use the details of the approximate eigenvector calculation, e.g., random walks and related diffusion-based methods, to understand viral diffusion problems. But in all those cases, we are interested in the leading nontrivial eigenvector of the Laplacian. We'll have a lot more to say about that later, but for now think of it just as some vector that in some sense describes important directions in the graph, in which case what this vanilla spectral algorithm does is putting or "embedding" the nodes of the graph on this line and cuts the nodes into two pieces, a left half and a right half. (The embedding has big distortion, in general, for some points at least; the two halves can be very unbalanced, etc.; but at least in very nice cases, that informal intuition is true, and it is true more generally if the two halves can be unbalanced, etc. Understanding these issues will be important for what we do.) (Make a picture on the board.)

We can ask: what is the optimization problem that this algorithm solves? As we will see eventually, in some sense, what makes a spectral graph algorithm a spectral graph algorithm is the first step, and so let's focus on that. Here is a basic spectral optimization problem that this problem solves.

$$\begin{aligned} \min \quad & x^T L_G x \\ \text{s.t. } & x^T D_G x = 1 \\ & x^T D_G \vec{1} = 0 \\ & x \in \mathbb{R}^V \end{aligned}$$

That is, find the vector that minimizes the quadratic form $x^T L x$ subject to the constraints that x sits on a (degree-weighted) unit ball and that x is perpendicular (in a degree-weighted norm) to the "trivial" all-ones vector. The solution to this problem is a vector, and it is the leading nontrivial eigenvector of L or $\mathcal{L}$.

Importantly, this is *not* a convex optimization problem; but rather remarkably it can be solved in low-degree polytime by computing an eigenvector of L. How can it be that this problem is solvable if it isn't convex? After all, the usual rule of thumb is that convex things are good and non-convex things are bad. There are two (related, certainly not inconsistent) answers to this.

- One reason is that this is an eigenvector (or generalized eigenvector) problem. In fact, it involves computing the leading nontrivial eigenvector of L, and so it is a particularly nice eigenvalue problem. And computing eigenvectors is a *relatively* easy thing to do—for example, with a black box solver, or in this special case with random walks. But more on this later.
- Another reason is that is is secretly convex, in that it is convex in a different place. Importantly, that different place there are better duality properties for this problem, and so it can be used to understand this problem and its solution better.

Both of these will be important, but let's start by focusing on the second reason. Consider the following version of the basic spectral optimization problem.

$$\begin{aligned} \mathsf{SDP}: \quad \min \quad & L \bullet X \\ \text{s.t.} \quad & \mathbf{Tr}(X) = I_0 \bullet X = 1 \\ & X \succeq 0, \end{aligned}$$

where $\bullet$ stands for the Trace, or matrix inner product, operation, *i.e.*, $A \bullet B = \mathbf{Tr}\left(AB^T\right) = \sum_{ij} A_{ij}B_{ij}$ for matrices A and B. *Note that, both here and below, I_0 is sometimes the Identity on the subspace perpendicular to the all-ones vector. This will be made more consistent later.* SDP is a relaxation of the spectral program SPECTRAL from an optimization over unit vectors to an optimization over distributions over unit vectors, represented by the density matrix X. But, the optimal values for SPECTRAL and SDP are the same, in the sense that they are given by the second eigenvector v of L for SPECTRAL and by $X = vv^T$ for SDP.

Thus, this is an SDP. While solving the vanilla spectral optimization problem with a black-box SDP solver is possible, it is not advisable, since one can just can a black-box eigenvalue solver (or run some non-black-box method that approximates the eigenvalue). Nevertheless, the SDP can be used to understand spectral graph methods. For example, we can consider the dual of this SDP:

$$\begin{aligned} \text{maximize} \quad & \alpha \\ \text{s.t.} \quad & L_G \succeq \alpha L_{K_n} \\ & \alpha \in \mathbb{R} \end{aligned}$$

This is a standard dual construction, the only nontrivial thing is we write out explicitly $I_0 = L_{K_n}$, as the identity matrix on the subspace perpendicular to $\vec{1}$ is $I_0 = I - \vec{1}^T\vec{1} = L_{K_n}$ where K_n is the complete graph on n vertices.

We will get into more detail later what this means, but informally this means that we are in some sense "embedding" the Laplacian of the graph in the Laplacian of a complete graph. Slightly less informally, the $\succeq$ is an inequality over graphs (we will define this in more detail later) which says that the Laplacian quadratic form of one graph is above or below that of another graph (in the sense of SPSD matrices, if you know what that means). So, in this dual, we want to choose the largest α such that that inequality is true.

1.7 Connections with random walks and diffusions

A final thing to note is that the vector x^* that solves these problems has a natural interpretation in terms of diffusions and random walks. This shouldn't be surprising, since one of the ways this vector is to partition a graph into two pieces that captures a qualitative notion of connectivity. The interpretation is that is you run a random walk—either a vanilla random walk defined by the matrix $W = D^{-1}A$ above, meaning that at each step you go to one of your neighbors with equal probability, or a fancier random walk—then x^* defines the slowest direction to mixing, i.e., the direction that is at the pre-asymptotic state before you get to the asymptotic uniform distribution.

So, that spectral graph methods are useful in these and other applications is largely due to two things.

- Eigenvectors tend to be "global" things, in that they optimize a global objective over the entire graph.
- Random walks and diffusions optimize almost the same things, but they often do it in a very different way.

One of the themes of what we will discuss is the connection between random walks and diffusions and eigenvector-based spectral graph methods on different types of graph-based data. Among

other things, this will help us to address local-global issues, e.g., the global objective that defines eigenvectors versus the local nature of diffusion updates.

Two things should be noted about diffusions.

- Diffusions are robust/regularized notions of eigenvectors
- The behavior of diffusions is very different on K_n or expander-like metric spaces than it is on line-like or low-dimensional metric spaces.

An important subtlety is that most data have some sort of degree heterogeneity, and so the extremal properties of expanders are mitigated since it is constant-degree expanders that are most unlike low-dimensional metric spaces. (In the limit, you have a star, and there it is trivial why you don't have good partitions, but we don't want to go to that limit.)

1.8 Low-dimensional and non-low-dimensional data

Now, complete graphs are very different than line graphs. So, the vanilla spectral graph algorithm is also putting the data in a complete graph. To get a bit more intuition as to what is going on and to how these methods will perform in real applications, consider a different type of graph known as an expander. I won't give the exact definition now—we will later—but expanders are very important, both in theory and in practice. (The former may be obvious, while the latter may be less obvious.) For now, there are three things you need to know about expanders, either constant-degree expanders and/or degree-heterogeneous expanders.

- Expanders are extremely sparse graphs that do not have any good clusters/partitions, in a precise sense that we will define later.
- Expanders are also the metric spaces that are least like low-dimensional spaces, e.g., a line graph, a two-dimensional grid, etc. That is, if your intuition comes from low-dimensional places like 1D or 2D places, then expanders are metric spaces that are most different than that.
- Expanders are sparse versions of the complete graph, in the sense that there are graph inequalities of the form $\succeq$ that relate the Laplacian quadratic forms of expanders and complete graphs.

So, in a certain precise sense, the vanilla spectral graph method above (as well as other non-vanilla spectral methods we will get to) put or embed the input data in two extremely different places, a line as well as a dense expander, i.e., a complete graph.

Now, real data have low-dimensional properties, e.g., sometimes you can visualize them in a two-dimensional piece of paper and see something meaningful, and since they often have noise, they also have expander-like properties. (If that connection isn't obvious, it soon will be.) We will see that the properties of spectral graph methods when applied to real data sometimes depend on one interpretation and sometimes depend on the other interpretation. Indeed, many of the properties—both the good properties as well as the bugs/features—of spectral graph methods can be understood in light of this tension between embedding the data in a low-dimensional place and embedding the data in an expander-like place.

There are some similarities between this—which is a statement about different types of graphs and metric spaces—and analogous statements about random vectors in $\mathbb{R}^n$, e.g., from a full-dimensional Gaussian distribution in $\mathbb{R}^n$. Some of these will be explored.

1.9 Small world and heavy-tailed examples

There are several types or classes of generative models that people consider, and different communities tend to adopt one or the other class. Spectral graph methods are applied to all of these, although they can be applied in somewhat different ways.

- Discretization or random geometric graph of some continuous low-dimensional place, e.g., a linear low-dimensional space, a low-dimensional curved manifold, etc. In this case, there is a natural low-dimensional geometry. (Put picture on board.)
- Stochastic blockmodels, where there are several different types of individuals, and each type interacts with individuals in the same group versus different groups with different probabilities. (Put picture on board, with different connection probabilities.)
- Small-world and heavy-tailed models. These are generative graph models, and they attempt to capture some local aspect of the data (in one case, that there is some low-dimensional geometry, and in the other case, that there is big variability in the local neighborhoods of individuals, as captured by degree or some other simple statistic) and some global aspect to the data (typically that there is a small diameter).

We will talk about all three of these in due course, but for now let's say just a bit about the small-world models and what spectral methods might reveal about them in light of the above comments. Small-world models start with a one-dimensional or two-dimensional geometry and add random edges in one of several ways. (Put picture here.) The idea here is that you reproduce local clustering and small diameters, which is a property that is observed empirically in many real networks. Importantly, for algorithm and statistical design, we have intuition about low-dimensional geometries; so let's talk about the second part: random graphs.

Consider G_{np} and G_{nm}, which are the simplest random graph models, and which have an expected or an exact number of edges, respectively. In particular, start with n isolated vertices/nodes; then:

- For G_{np}, insert each of the $\binom{n}{2}$ possible edges, independently, each with probability p.
- For G_{nm}, among all $\binom{\binom{n}{2}}{m}$ subsets of m edges, select one, independently at random.

In addition to being of theoretical interest, these models are used in all sorts of places. For example, they are the building blocks of stochastic block models, in addition to providing the foundation for common generative network models. (That is, there are heavy-tailed versions of this basic model as well as other extensions, some of which we will consider, but for now let's stick with this.) These vanilla ER graphs are often presented as strawmen, which in some sense they are; but when taken with a grain of salt they can reveal a lot about data and algorithms and the relationship between the two.

First, let's focus on G_{np}. There are four regimes of particular interest.

- $p < \frac{1}{n}$. Here, the graph G is not fully-connected, and it doesn't even have a giant component, so it consists of just a bunch of small things.
- $\frac{1}{n} \lesssim p \lesssim \frac{\log(n)}{n}$. Here there is a giant component, i.e., set of $\Omega(n)$ nodes that are connected, that has a small $O(\log(n))$ diameter. In addition, random walks mix in $O(\log^2(n))$ steps, and the graph is locally tree-like.
- $\frac{\log(n)}{n} \lesssim p$. Here the graph is fully-connected. In addition, it has a small $O(\log(n))$ diameter and random walks mix in $O(\log^2(n))$ steps (but for a slightly different reason that we will get back to later).
- $\frac{\log(n)}{n} \ll p$. Here the graph is pretty dense, and methods that are applicable to pretty dense graphs are appropriate.

If $p \gtrsim \log(n)/n$, then G_{np} and G_{nm} are "equivalent" in a certain sense. But if they are extremely sparse, e.g., $p = 3/n$ or $p = 10/n$, and the corresponding values of m, then they are different.

In particular, if $p = 3/n$, then the graph is not fully-connected, but we can ask for a random r-regular graph, where r is some fixed small integer. That is, fix the number of edges to be r, so we have a total of nr edges which is *almost* a member of G_{nm}, and look at a random such graph.

- A random 1-regular graph is a matching.
- A random 2-regular graph is a disjoint union of cycles.
- A random 3-regular graph: it is fully-connected and had a small $O(\log(n))$ diameter; it is an expander; it contains a perfect matching (a matching, i.e., a set of pairwise non-adjacent edges that matches all vertices of the graph) and a Hamiltonian cycle (a closed loop that visits each vertex exactly once).
- A random 4-regular graph is more complicated to analyze

How does this relate to small world models? Well, let's start with a ring graph (a very simple version of a low-dimensional lattice, in which each node is connected to neighbors within a distance $k = 1$) and add a matching (which is a bunch of random edges in a nice analyzable way). Recall that a random 3-regular graph has both a Hamiltonian cycle and a perfect matching; well it's also the case that the union of an n cycle and a random machine is contiguous to a random 3 regular random graphs. (This is a type of graph decomposition we won't go into.)

This is a particular theoretical form to say that small world models have a local geometry but globally are also expanders in a strong sense of the word. Thus, in particular, when one runs things like diffusions on them, or relatedly when one runs spectral graph algorithms (which have strong connections under the hood to diffusions) on them, what one gets will depend sensitively on the interplay between the line/low-dimensional properties and the noise/expander-like properties.

It is well known that similar results hold for heavy-tailed network models such as PA models or PLRG models or many real-world networks. There there is degree heterogeneity, and this can give a lack of measure concentration that is analogous to the extremely sparse Erdos-Renyi graphs, unless one does things like make minimum degree assumptions. It it less well known that similar things also hold for various types of constructed graphs. Clearly, this might happen if one constructs stochastic blockmodels, since then each piece is a random graph and we are interested in the

interactions between different pieces. But what if construct a manifold method, but there is a bit of noise? This is an empirical question; but noise, if it is noise, can be thought of as a random process, and in the same way as the low-dimensional geometry of the vanilla small world model is not too robust to adding noise, similarly geometric manifold-based methods are also not too robust.

In all of this, there algorithm questions, as well as statistical and machine learning questions such as model selection questions and questions about how to do inference with vector-based of graph-based data, as well as mathematical questions, as well as questions about how these methods perform in practice. We will revisit many of these over the course of the semester.

1.10 Outline of class

In light of all that, here is an outline of some representative topics that we will cover.

1. Basics of graph partitioning, including spectral, flow, etc., degree heterogeneity, and other related objectives.
2. Connections with diffusions and random walks, including connections with resistor network, diffusion-based distances, expanders, etc.
3. Clustering, prediction, ranking/centrality, and communities, i.e., solving a range of statistics and data analysis methods with variants of spectral methods.
4. Graph construction and empirical properties, i.e., different ways graphs can be constructed and empirical aspects of "given" and "constructed" graphs.
5. Machine learning and statistical approaches and uses, e.g., stochastic blockmodels, manifold methods, regularized Laplacian methods, etc.
6. Computations, e.g., nearly linear time Laplacian solvers and graph algorithms in the language of linear algebra.