Understanding Network Connectivity

A Linear Algebra Approach to Graph Theory

Joey Bejjani and Harvey Lin

December 12, 2022

Cover Letter

We would like to thank our peer reviewers Ethan Tan, Dai Pham, Jennifer Song, Finn Jensen, Richard Calvo, and Pema Choedon, as well as our project reader Eunice Sukarto for their feedback on the draft of our paper. One piece of advice offered by several of our reviewers was to break up longer proofs into smaller sections for readability. We split up the proofs of Lemmas 1 and 4 into separate paragraphs and kept their advice in mind when writing the proof of Theorem 1 for the final draft.

Another common suggestion was to restructure the flow of Section 4.2, in which we present a sequence of lemmas necessary for a proof of Theorem 2. Previously, we had left the statement of Theorem 2 to the end of the section right before the proof. The reviewers recommended we instead state where we are taking the reader at the beginning of the section, and then take them there step by step. In line with this suggestion, we moved the statement of Theorem 2 to the beginning of the section, added transitions between the lemmas reminding the reader where they are headed, then reiterated the claim of Theorem 2 before proving it at the end of the section.

Following another suggestion, we added a 'roadmap' paragraph at the end of our introduction to outline how the rest of the paper will proceed. We were also advised to remind the reader of our central question throughout the paper. We did this at the beginning of Sections 4.1 and 5. Our reviewers also suggested we make the motivation of each section clearer and more closely tie our discussion of matrices back to the overarching theme of network analysis. We did this by reminding the reader of the motivation of the adjacency matrix in Section 3 and discussing its use in examining network density. We then give a clearer transition to the Laplacian matrix, which we see builds off of the adjacency matrix. Throughout the final draft we also wove in more examples of applications of graph theory to real-world networks as well as directions for further study. We hope that this sparks the reader's interest in continuing to study graphs as a tool for solving network-related problems.

To help accomplish this goal we also added Section 7, which applies all of the linear algebra tools discussed in the paper in actually analyzing a real-world network. In this section we also point the reader to `www.konect.cc`, which is home to a variety of interesting network datasets, and include Python code snippets to help the reader get started analyzing larger networks on their own.

Joey completed sections 1, 4, and 5, and Harvey completed sections 2, 3, and 6, though we both made meaningful contributions to all sections. Section 7 was completed together. Thank you for reading our paper.

Contents

1 Introduction

Networks are the framework for much of the modern world because they are built on connectivity. Consider computer networks, road networks, social networks, brain networks, etc.—these all rely on the linking of “nodes,” whether those are computers, cities, people, neurons. The motivation of this paper is to explore the ways in which we can use linear algebra to analyze important properties of real-world networks, which can be usefully represented as graphs. We will see that graph theory, then, is a powerful tool for understanding and solving connectivity-related problems for a given network.

What are some real-world connectivity problems that motivate our study of graph theory? Graphs can be used to design transport networks or to optimise routes for delivering mail within a city, for example. The classic ‘travelling salesman’ problem is derived from this idea and can be solved by representing it as a graph. Companies like Facebook and LinkedIn model users and their degree of connection to other users with graphs to decide who to recommend to add as a new friend or connection or to find two users’ mutual friends. Sports tournaments can also be modelled as graphs, with connections between nodes representing two teams playing each other along with the corresponding result.

What role does linear algebra play in studying graphs and solving problems like these? We will see how matrix representation of graphs captures certain essential characteristics of graphs, enabling us to better understand their structure and behavior. The main focus of the project will be the Laplacian matrix representation of graphs in particular. The unique properties of Laplacian matrices will help us answer the following central question: **how can we gain insight into a graph’s connectivity?** And perhaps more importantly, **what can we do with this insight?** By giving us a window into connectivity, Laplacian matrices can help us in analyzing the security of a computer network [Wal19], for example, or even to understand Alzheimer’s disease in terms of network breakdown in the brain [Dai14].

The paper will proceed as follows. First, we will formally define *graph* and important terms used in graph theory to lay the groundwork for the following sections. We will then explore the adjacency matrix as a preliminary tool for representing and understanding graphs. Building off the adjacency matrix, we will define the Laplacian matrix and use it to analyze graph connectivity. We will then use our new insight into connectivity to see how we can break up, or partition, a graph, as well as rank the importance of nodes in a graph. Finally, we will apply all of these tools in analyzing the collaborative network of jazz musicians using a real dataset.

2 Properties of Graphs

As we begin to understand the role of graph theory in network analysis, the first question we ask is, **what is a graph?** What are the properties of graphs and how might they encode different features of a network?

Definition 1

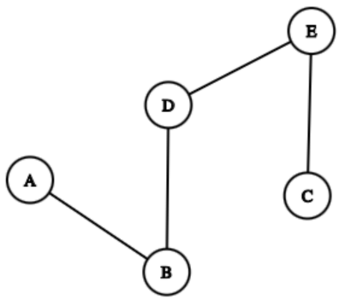
A **graph** G is an object consisting of two sets: the *vertex set*, $V(G)$, and the *edge set*, $E(G)$. The vertex set is a finite nonempty set. The edge set may be empty (say, when no vertices are connected), but otherwise its elements are two-element subsets of the vertex set.

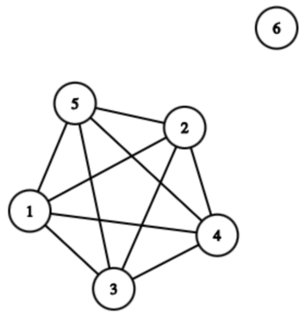
The elements of $V(G)$ are called *vertices* and the elements of $E(G)$ are called *edges*.

Note that a vertex can also be referred to as a *node* or a *point*.

[Tru94]

Let us consider two examples. Let G be a graph with vertex set $\{A, B, C, D, E\}$ and edge set $\{\{A, B\}, \{B, D\}, \{C, E\}, \{D, E\}\}$. Let H be a graph with vertex set $\{1, 2, 3, 4, 5, 6\}$ and edge set $\{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$. We can visualize these two graphs by drawing points, representing vertices, connected by lines, representing edges. G is drawn on the left and H on the right:





Definition 2

For a graph G, vertices X and Y are **adjacent** if there is an edge $\{X, Y\} \in E(G)$. We say that $\{X, Y\}$ joins or connects X and Y.

The edge $\{X, Y\}$ is *incident* to each of X and Y, and each of X and Y is incident to $\{X, Y\}$.

Two edges incident to the same vertex are called *adjacent edges*. A vertex incident to no edges at all is *isolated*.

[Tru94]

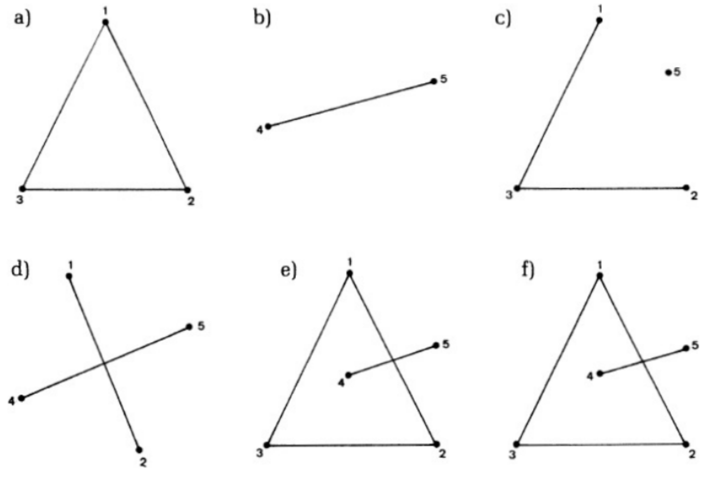
In our graph G, for example, each of A and B is incident to $\{A, B\}$, so A and B are adjacent. However, A and C are not adjacent, since $\{A, C\} \notin E(G)$. Also, edges $\{A, B\}$ and $\{B, D\}$ are adjacent, since they are both incident to vertex B. Moreover, vertex 6 in graph H is isolated.

Definition 3

Graph H is a **subgraph** of graph G if $V(H) \subset V(G)$ and $E(H) \subset E(G)$.

[Tru94]

For example, in the figure below, graphs a through e are subgraphs of f:



[Tru94]

Definition 4

The **degree** of a vertex is the number of edges incident to it.

Moreover, a graph is ***d*-regular** if every vertex has degree d.

[Tru94]

In graph G, vertex A has degree 1, B has degree 2, C has degree 1, D has degree 2, and E has degree 2. So, G is clearly not a d-regular graph, but a, b, and d in the above figure are 2-degree, 1-degree, and 1-degree graphs respectively.

So far, we have talked about graphs without any notion of direction along edges between the vertices. We could not say, for example, that edge $\{1, 2\}$ in H 'goes' from vertex 1 to vertex 2; after all, sets do not 'care' about order, so $\{1, 2\}$ is equivalent to $\{2, 1\}$. So really, our previous definition for graph is a definition for an **undirected graph**. How might we then define a directed graph?

Definition 5

A **directed graph** G is a graph in which the edge set consists of ordered pairs of elements in the vertex set.

Edge $(1, 2)$ in a directed graph could then be represented visually as an arrow pointing from vertex 1 to vertex 2. This definition is included as a stepping stone for further study in graph theory; our paper will focus on undirected graphs.

Recall that the motivation of the paper is to analyze connectivity in a graph. The following definitions are then necessary:

Definition 6

A **walk** in a graph is a sequence $A_1A_2A_3 \dots A_n$ of not necessarily distinct vertices in which A_1 is joined by an edge to A_2, A_2 is joined by an edge to A_3, ..., and A_{n-1} is joined by an edge to A_n. The walk $A_1A_2A_3 \dots A_n$ is said to join A_1 and A_n.

[Tru94]

In graph H, for example, we can observe that there are two walks of length 2 joining vertices 1 and 4: 1, 3, 4 and 1, 2, 4. (What are the other walks joining 1 and 4? What are their lengths?) As another example, in G, there is one walk of length 3 joining vertices B and C.

Definition 7

A graph is **connected** if every pair of vertices is joined by a walk. Otherwise, the graph is disconnected.

[Tru94]

Observe that G is connected, while H is disconnected. Finally, consider the following definition to round off the section:

Definition 8

A **connected component** of a graph is a connected subgraph that is not contained in a larger connected subgraph.

[Tru94]

Thus a connected graph is its own single connected component, and the connected components of a disconnected graph are what we see visually as the "pieces" comprising it.

3 The Adjacency Matrix

While drawing graphs is a very useful way to identify global structure, how might we uncover the less visible characteristics of a graph? Naturally, we would like to represent graphs in such a way that we can manipulate them quickly and perform computations. Linear algebra gives us the tools for this. Namely, we can use matrices to represent and analyze graphs mathematically. First, let us look at the adjacency matrix of a graph, which gives us a useful starting point for analyzing a graph's properties:

Definition 9

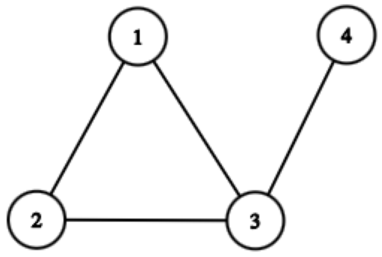
Suppose G is a graph with $V(G) = \{1, ..., n\}$. The **adjacency matrix** of G, denoted $A(G)$, is the n x n matrix whose rows and columns are indexed by $V(G)$ and whose (i, j)-th entry is defined by the following:

$$A_{ij} = \begin{cases} 1 & \text{if } i \neq j \text{ and the vertices } v_i, v_j \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}$$

[Bap14]

Note that adjacency matrices have zeroes along the main diagonal, since these represent entries where $i = j$. The adjacency matrix of an undirected graph is also symmetric, since v_i adjacent to v_j implies v_j adjacent to v_i.

Let us try constructing an adjacency matrix following our definition. Consider the following graph, G_1, with $V(G_1) = \{1, 2, 3, 4\}$:



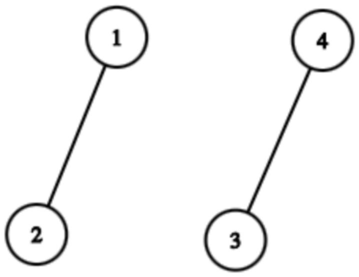
This has the corresponding 4 x 4 adjacency matrix:

$$A(G_1) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

We see, for instance, that entry $(2, 1)$ is 1, since vertex 2 is adjacent to vertex 1. By symmetry, entry $(1, 2)$ is also 1. Since there is no edge between vertices

1 and 4, we see corresponding 0s in the adjacency matrix.

Consider another example, which we label G_2. This is a disconnected graph:



Constructing the adjacency matrix again, we have:

$$A(G_2) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Observe that $A(G_2)$ is block diagonal.

Definition 10

A **block diagonal** matrix is a square diagonal matrix in which the diagonal elements are square matrices of any size and off-diagonal elements are 0. Each of the square matrices along the diagonal is called a *block*.

With this definition in mind, we can concisely write the adjacency matrix as

$$A(G_2) = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \text{ with blocks } A_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \text{ and } A_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Observe that the two blocks represent the two connected components of G_2. All values outside the blocks are zeroes because there are no edges between connected components.

In general, for any disconnected graph, we can write its adjacency matrix as block diagonal with each block representing a connected component of the graph. Hence, the number of connected components in the graph is the number of blocks in the block diagonal adjacency matrix. [Mar13].

3.1 Calculating Walks With the Adjacency Matrix

An elegant and powerful result about the number of walks in a graph can be calculated quite easily by taking powers of the associated adjacency matrix. Specifically,

Theorem 1

Let G be a graph with $V(G) = \{v_1, v_2, ..., v_n\}$ and adjacency matrix $A(G) = A$. Then, the (i, j)th entry of A^k is the number of walks from v_i to v_j of length k.

Proof: We prove by mathematical induction on k. Denote the (i, j)th entry of A by $A(i, j)$.

Base Case: Suppose $k = 1$. Then, $A(i, j)$ can be 1 or 0 by the definition of adjacency matrix.

- $A(i, j) = 1 \iff v_i$ is adjacent to $v_j \iff$ there is one walk of length $k = 1$ from v_i to v_j.
- $A(i, j) = 0 \iff v_i$ is not adjacent to $v_j \iff$ there is no walk of length $k = 1$ from v_i to v_j.

So, the theorem holds in the base case.

Inductive Step: Our inductive hypothesis is the following: for some $k \geq 1$, the (i, j)th entry of A^k is the number of walks from v_i to v_j of length k.

Now, consider the number of walks of length $k+1$ from v_i to v_j. Notice that any walk of length $k+1$ from v_i to v_j must first be a walk of length k from v_i to some v_h, such that v_h is adjacent to v_j. Then, to count the total number of walks of length $k+1$, we need to sum the product of walks from v_i to v_h with walks from v_h to v_j, over all allowed v_h. Mathematically, this is:

$$\textstyle\sum_{v_h \in N(v_j)} (\text{number of walks of length } k \text{ from } v_i \text{ to } v_h).$$

(*Remark:* $N(v_j)$ denotes the neighboring vertices of v_j; that is, the set of vertices that are adjacent to v_j.)

By the inductive hypothesis, this is

$$\textstyle\sum_{v_h \in N(v_j)} A^k(i, h) = \sum_{h=1}^{n} [A^k(i, h)][A(h, j)].$$

And by the definition of matrix multiplication, this is just

$$A^{k+1}(i, j)$$

or the $(k+1)$-st case. From the base case and inductive step, our claim then holds by induction on k. ■

Exercise: Using this result, verify our observation in Section 2 that graph H has two walks of length 2 from vertex 1 to vertex 4.

With a concrete way of calculating the number of walks between a pair of vertices, we can determine how ‘dense’ a particular network or graph is by comparing the number of walks with the number of vertices.

The density of a network is highly related to its connectivity. For example, consider a graph representing the network of employees in a company, where vertices represent employees and edges meaningful interactions between them. If the graph has low density, this might suggest a problem with information flow in the company, perhaps with instructions from upper-level management reaching lower-level workers. Moreover, a highly dense subway network would indicate that if a fault on a particular section of track occurred, commuters could still find reasonable alternative routes, cushioning the possible disruption in the network.

From this result, we see how the adjacency matrix gives us a way to analyze the relationship between pairs of vertices and draw conclusions about the density of a graph. In the following sections, we will build off our work on the adjacency matrix to look at another, related way of representing a graph: the Laplacian matrix. While the adjacency matrix tells us about the relationship between vertices, the Laplacian matrix gives us a direct window into network connectivity, our paper’s central motivation.

4 The Laplacian Matrix

Definition 11

Let G be a graph with n vertices. The **Laplacian matrix** of G, denoted $L(G)$, is the n x n matrix whose (i, j)-th entry is defined by the following:

$$L_{ij} = \begin{cases} -1 & \text{if } i \neq j \text{ and vertices } v_i, v_j \text{ are adjacent} \\ 0 & \text{if } i \neq j \text{ and vertices } v_i, v_j \text{ are not adjacent} \\ d_i & \text{if } i = j \end{cases}$$

where d_i denotes the degree of vertex v_i.

Let $D(G)$ be the diagonal matrix whose entries are the degrees of each vertex of G, and let $A(G)$ be the adjacency matrix of G. Then we can also define the Laplacian matrix as follows:

$$L(G) = D(G) - A(G) \tag{1}$$

[Joh17]

Note that like the adjacency matrix, the Laplacian matrix of an undirected graph is symmetric. (Why?)

Let us construct the Laplacian matrix of graph G_1 from our earlier example. We get:

$$L(G_1) = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

We also have:

$$D(G_1) = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now we verify Equation (1) using $A(G_1)$ from earlier:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} = L(G_1)$$

4.1 The Laplacian Spectrum

We now have the necessary groundwork to answer our central question: **how can we gain insight into a graph's connectivity, and what can we do with this insight?** In the rest of Section 4, we will see how Laplacian matrices reveal information about the connectivity of a graph.

The **spectrum** of a matrix is the set of its eigenvalues and their multiplicities. By examining the spectrum of the Laplacian matrix of a graph, we can learn whether the graph is connected and how well it is connected.

To ground our discussion with examples, we also return to our previous disconnected graph G_2, and calculate $L(G_2)$, a result we will need for later. We have:

$$L(G_2) = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

Observe that this matrix, like $A(G_2)$, is block diagonal.

4.2 The Smallest Eigenvalue

We will now present a sequence of lemmas building to a proof of Theorem 2, which gives us a tool for analyzing network connectivity using the Laplacian spectrum. For simplicity, in Lemmas 1-4 and Theorem 2, let G be an undirected unweighted d-regular graph. (Proofs outside our scope exist for graphs that are not d-regular.)

Theorem 2

The number of connected components of graph G is equal to the algebraic multiplicity of eigenvalue 0 of $L(G)$.

We first need to show that 0 is an eigenvalue of $L(G)$. In fact,

Lemma 1

The smallest eigenvalue of $L(G)$ is always zero. That is, $\lambda_1 = 0$.

[Mar13]

Proof: From Lemma 7 of [Kotnd], $L(G)$ is positive semidefinite, meaning its eigenvalues are non-negative. Thus, $\lambda_1 \geq 0$.

We now show that 0 is an eigenvalue of $L(G)$. Note that the sum of the entries in row i of $A(G)$ gives the degree of vertex i. Since G is d-regular, this sum is equal to d for each row i.

$$\text{Then, letting } \vec{v} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \text{ we have } A(G) \cdot \vec{v} = \begin{bmatrix} d \\ d \\ \vdots \\ d \end{bmatrix}.$$

$$\text{Observing that } D(G) \cdot \vec{u} = \begin{bmatrix} d \\ d \\ \vdots \\ d \end{bmatrix}, \text{ we conclude,}$$

$$(D(G) - A(G)) \cdot \vec{v} = \begin{bmatrix} d \\ d \\ \vdots \\ d \end{bmatrix} - \begin{bmatrix} d \\ d \\ \vdots \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 0 \cdot \vec{v}.$$

Thus, 0 is an eigenvalue of $D(G) - A(G) = L(G)$. Therefore, $\lambda_1 = 0$. ■

We will need the following lemma:

Lemma 2

Let $x \in \mathbb{R}^n$ where $x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$. Let L be the Laplacian matrix of graph G and $E(G)$ be the set of edges $\{i, j\}$ in G. Then,

$$x^T L x = \sum_{[i,j] \in E(G)} (x_i - x_j)^2.$$

See Lemma 3.6 of [Mar13] for a proof.

as well as this more intuitive lemma relating the determinant of a block diagonal matrix to the determinants of each block:

Lemma 3

Let B be a block diagonal matrix with blocks $A_1, A_2, ..., A_n$. Then, $det B = det A_1 det A_2 ... det A_n$.

[Mar13]

We now prove Lemma 4, which is just the case of Theorem 2 where eigenvalue 0 has multiplicity 1:

Lemma 4

The algebraic multiplicity of eigenvalue 0 of $L(G)$ is equal to 1 if and only if G is connected.

Proof: We prove the forward direction by its contrapositive; i.e. if G is not connected, the algebraic multiplicity of λ_1 is greater than 1.

Supposing G is not connected, we can write $A(G)$ as a block diagonal matrix. We can do this by labelling vertices contained in the same connected component of G consecutively: vertices $V_1 = \{1, 2, ..., a-1, a\}$ are part of one connected component, vertices $V_2 = \{a+1, a+2, ..., a+b-1, a+b\}$ are part of another connected component, and so on.

Let A_1 be the adjacency matrix for the connected component with V_1, A_2 the adjacency matrix for the connected component with V_2, and so on. Adjacency matrix $A(G)$ can then be written as

$$\begin{bmatrix} A_1 & & \dots & 0 \\ & A_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & & A_n \end{bmatrix}$$

where all values outside the A_i blocks are 0, since there are no edges between the connected components.

Similarly, the Laplacian matrix of G is block diagonal (as we saw in the $L(G_2)$ example). We can quickly see this from Equation (1): since $D(G)$ and $A(G)$ are block diagonal, it follows that their difference, $L(G)$, is also block diagonal.

Let the blocks in $L(G)$ be denoted by $L_1, ..., L_n$. By Lemma 1, each block in $L(G)$ has eigenvalue 0. Hence, for all i, $det(L_i - \lambda I)$ has an $(x - 0)$ term by the definition of eigenvalue. By Lemma 3,

$$det(L(G) - \lambda I) = det(L_1 - \lambda I)...det(L_n - \lambda I).$$

Then, $det(L(G) - \lambda I)$ has more than one $(x - 0)$ term, so the algebraic multiplicity of $\lambda_1 = 0$ for $L(G)$ is greater than 1.

We prove the reverse direction by contradiction. Suppose that G is connected, but the algebraic multiplicity of eigenvalue 0 is greater than 1.

By Corollary 2.13 from [Mar13], there then exist at least two linearly independent eigenvectors v_1 and v_2, with eigenvalue 0. Then, $v_1^T L v_1 = 0$ and $v_2^T L v_2 = 0$. By Lemma 2, we have

$$\sum_{\{i,j\} \in E(G)} (v_{i1} - v_{j1})^2 = \sum_{\{i,j\} \in E(G)} (v_{i2} - v_{j2})^2 = 0.$$

Hence, for all $\{i, j\} \in E(G)$, we have $v_{i1} = v_{j1}$ and $v_{i2} = v_{j2}$. Note that if there is a path between two vertices a and b, then $v_{a1} = v_{b1}$ and $v_{a2} = v_{a2}$ even if $\{i, j\} \notin E(G)$. Since G is connected, there is a path between any pair of vertices. Thus, v_1 and v_2 must be constant vectors (that is, the entries of v_1 are the same, and the entries of v_2 are the same). Then, $v_1 = cv_2$ for some scalar c, contradicting the linear independence of v_1 and v_2. ■
[Mar13]

Observing $L(G_2)$, we suspect that it has more than one zero eigenvalue. We ask, what does the algebraic multiplicity of λ_1 tell us about a graph? Theorem 2, which we can now prove, gives us the answer.

Claim: The number of connected components of graph G is equal to the algebraic multiplicity of eigenvalue 0 of $L(G)$.

Proof: As previously shown, G has n connected components $\iff A(G)$ can be block diagonal with n blocks $\iff L(G)$ has n blocks.

Let $L_1, ..., L_n$ be the blocks of $L(G)$. By Lemma 1, each block has eigenvalue 0, and by Lemma 4, the eigenvalue 0 of each block has multiplicity 1. By Lemma 3, $det(L(G) - \lambda I) = det(L_1 - \lambda I)...det(L_n - \lambda I)$. Then, the algebraic multiplicity of 0 for $L(G)$ is the sum of the algebraic multiplicity of 0 of each block L_i. If G has n connected components, this sum is $1 + \ldots + 1$, n times, or just n. ■
[Mar13]

Let us return to our example with $L(G_2)$ to show Theorem 2 in action. We find the eigenvalues of $L(G_2)$ by setting its characteristic polynomial equal to 0:

$$\rho_{L(G_2)}(\lambda) = det(L(G_2) - \lambda I) = \lambda^2(2 - \lambda)^2$$

So, $L(G_2)$ has an eigenvalue of 0 with algebraic multiplicity 2 and an eigenvalue of 2 with algebraic multiplicity 2. By Theorem 2, G_2 then has 2 connected components. We can confirm this with the diagram of G_2 from earlier, and by observing that $L(G_2)$ is block diagonal with 2 blocks.

Exercise: Without actually calculating the eigenvalues, what is the smallest eigenvalue and its algebraic multiplicity for the Laplacian of G_1 from earlier? How do you know?

4.3 The Second Smallest Eigenvalue

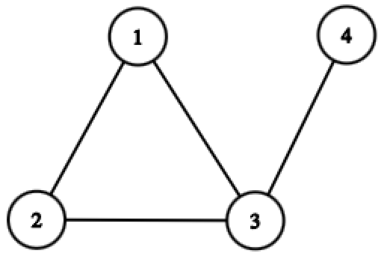
Counting repeated eigenvalues separately, we see from Lemma 4 that a graph G is connected if and only if the second smallest eigenvalue, λ_2, of $L(G)$ is greater than 0. (If G is disconnected, it has at least 2 connected components, and so the smallest and second smallest eigenvalues must be the same: $\lambda_1 = \lambda_2 = 0$.)

Given a connected graph, does the magnitude of λ_2 tell us anything interesting about the graph?

It turns out that λ_2 gives us a measure of how "well connected" G is. The further λ_2 is from 0, the more connected the graph.

Let us define λ_2 of $L(G)$ as the **algebraic connectivity** of G. This is also known as the **Fiedler Value** of G. Consider the following example of how we can compare the connectivity of graphs using the Fiedler Value.

Let us revisit G_1:



We found that

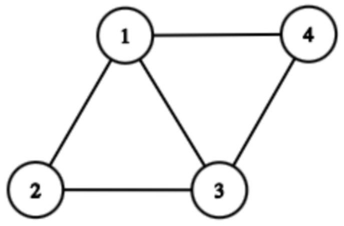
$$L(G_1) = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

The eigenvalues in order are

$$0, 1, 3, 4$$

Since G_1 is connected, we see that eigenvalue 0 has multiplicity 1. Hence, $\lambda_2 > 0$. In particular, we have $\lambda_2 = 1$, so our Fiedler Value is 1.

Now consider a new graph, G_3:



Just by looking at the diagram, G_3 seems to be more connected than G_1. Let's formalize this comparison.

The Laplacian is

$$L(G_3) = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

with eigenvalues

$$0, 2, 4, 4$$

The Fiedler Value of G_3, given by λ_2, is then 2. As expected, the Fiedler Value of G_3 is greater than the Fiedler Value of G_1. This indicates that G_3 is more robustly connected than G_1. This is intuitive: G_3 was simply obtained by adding an edge between vertices 1 and 4 in G_1.

What exactly does this comparison tell us? One takeaway is that G_3 is harder to "tear apart" than G_1; we would have to remove more edges from G_3 to break it up (i.e. disconnect the graph, making $\lambda_2 = 0$) than we would have to remove from G_1. What might this mean in the context of networks?

Connectivity is one measure of network security. If the graphs represented computer networks, for example, we might conclude that the G_3 network is more secure than the G_1 network. In an insecure computer network (one with low connectivity), one could simply remove a few nodes (computers) or edges (connections between computers) to disrupt information flow within the network. In fact, specialists have found ways to increase the security of a network by maximizing its Fiedler Value using semi-definite programming [Wal19].

As an area for future exploration, note that the Fiedler Value also determines the conductance of a graph, another measure of how "well-knit" a graph. Conductance has applications in electrical networks and circuits [Kotnd].

5 Graph Partitioning

We have answered the first part of our central question; by examining the Laplacian spectrum, we have gained insight into network connectivity by finding the number of connected components in a graph and comparing Fiedler Values. The second part remains: **how can we use this insight?** In this section, we will look at one way we can use our knowledge of graph connectivity to solve network-related problems. In particular, we mentioned "breaking up" a graph in the previous section, and how this task is easier for graphs with lower Fiedler Values. Using our new insight into connectivity, how might we actually go about breaking up a graph?

First, consider why we might want to break up a graph at all. When networks get very large, it becomes difficult for one computer to build and analyze its associated graph in memory. It becomes very useful to divide the graph into smaller pieces to make analysis more efficient. So, in studying networks, we often want to "partition" a graph, or "cluster" the vertices into different groups in a reasonable way.

In this paper, by a reasonable partitioning we mean removing as few edges as possible (graph theorists call this a *minimum cut*) while keeping the resulting subgraphs approximately equal in size. Let us see how we might achieve this partitioning goal.

5.1 The Fiedler Method

The Fiedler Method is one simple yet powerful way to partition a graph based on the connectivity of the vertices. Though it is not perfect—it does not always yield the "best" partitioning—various adjustments that we will not get into easily account for problems that might arise [WG21].

The Fiedler Method uses the Fiedler Vector of a graph:

Definition 12

The **Fiedler Vector** of a graph G is the eigenvector corresponding to the Fiedler Value.

According to the Fiedler Method, we can achieve a reasonable partition into two subgraphs by grouping the vertices of a graph according to the sign of the entries in the Fiedler Vector, where each entry corresponds to a vertex.

Let us outline the steps of the Fiedler Method:

1. For a connected graph $G = (V, E)$, construct the Laplacian matrix $L(G)$.
2. Compute the eigenvector, $\vec{v}$, corresponding to the first non-trivial eigenvalue of $L(G)$. This is the Fiedler Vector.
3. Let v_i indicate the ith entry of the Fiedler Vector. For each vertex $i \in V$,

a. if $v_i > 0$, put vertex i in partition $V^{(+)}$
b. if $v_i < 0$, put vertex i in partition $V^{(-)}$
c. if $v_i = 0$, we must make a choice of where to put vertex i

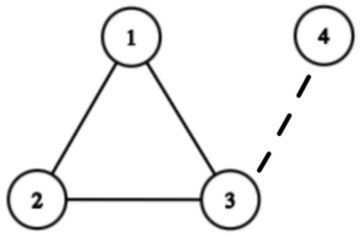
For a complete justification for why the Fiedler Method works, see [WG21].

Let us try partitioning G_1 using the Fiedler Method. Using $L(G_1)$, we found that the Fiedler Value of G_1 is 1. We get the corresponding Fiedler Vector, $\vec{v}$:

$$\vec{v} = \begin{bmatrix} -1 \\ -1 \\ 0 \\ 2 \end{bmatrix}$$

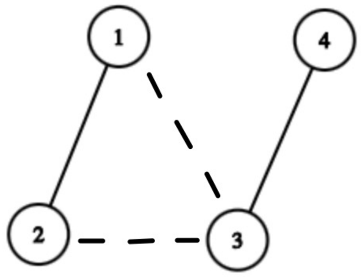
It can be easily checked that $\vec{v}$ is an eigenvector of $L(G_1)$ corresponding to eigenvalue $\lambda_2 = 1$.

We then group the vertices according to the signs of their corresponding entry in $\vec{v}$. The Fiedler Method does not explicitly say what to do with vertex 3, since $v_3 = 0$. We can partition either as $(\{1,2,3\},\{4\})$ or as $(\{1,2\},\{3,4\})$. In the first option, we only have to remove a single edge, represented by the dashed line:



From a balance point of view, this result is not very satisfactory, as we are left with a triangle on the left and a single vertex on the right.

Our second option requires removing two edges, but it yields balanced subgraphs, which is preferable in many applications. The partitioning $(\{1,2\},\{3,4\})$ gives G_2':



When applied to larger, more complex graphs, the Fiedler Method becomes a powerful way to see which groups of vertices in a graph are most connected. Partitioning a graph then not only lets us allocate computational resources more efficiently, it can help reveal the underlying structure of a network, breaking it up into its most connected sub-networks [WG21].

6 Laplacian Centrality

Building off our work in the previous section, we turn towards centrality in graphs. Our discussion of centrality is motivated by the question of which vertices in a network are most critical or important and has applications such as determining the most influential person(s) in a social network, key infrastructure nodes in the Internet or urban networks, super-spreaders of disease, and hubs in the human brain [vdH13]. There are numerous methods of calculating centrality. (Perhaps the simplest and most obvious is to rank centrality of vertices based on each vertex's degree—this is known as *degree centrality.*) The method we present in this section, **Laplacian centrality**, uses the now familiar Laplacian matrix of a graph. The idea behind Laplacian centrality is to assess the impact of a vertex being *deactivated*, or removed, from a particular graph. How a network responds to the removal of a node can then help us determine how 'central' that node is to the performance of the network. The discussion below will largely follow that of Xingqin Qi et al. in [ea11]; though Xingqin Qi et al. generalize their work to weighted graphs, we will only consider unweighted graphs.

6.1 Laplacian Energy and Laplacian Centrality Formula

Recall Definition 11, for the Laplacian matrix of a graph G with n vertices. Note that all following graphs in this section will consist of n vertices.

To calculate Laplacian centrality, we first need to understand Laplacian energy:

Definition 13

Let $\lambda_1, \lambda_2, ..., \lambda_n$ be the eigenvalues of $L(G)$. Then the **Laplacian energy** of G, or $E_L(G)$, is defined by

$$E_L(G) = \textstyle\sum_{i=1}^{n} \lambda_i^2$$

Practically, a formula for Laplacian energy in terms of d_i rather than λ_i may be more convenient, letting us avoid calculating the eigenvalues. In fact, we know that

$$E_L(G) = \sum_{i=1}^{n} \lambda_i^2 = 2|E(G)| \tag{2}$$

[IG05]

Be careful to distinguish $E_L(G)$, the Laplacian energy of G, from $E(G)$, the edge set of G.

The Laplacian energy of G is then just twice the number of edges in G. We begin to see the relationship between Laplacian energy and connectivity: we increase a graph's Laplacian energy when we add more edges between vertices, which intuitively makes a graph more well-connected.

As a side remark, a neat formula for $E_L(G)$ for *weighted* graphs in terms of degrees d_i and weights w_i can also be found in [ea11].

We can now define Laplacian centrality:

Definition 14

Let G_i be the subgraph obtained by removing vertex v_i from a graph G (thus also removing edges previously incident to v_i). The **Laplacian centrality**, $C_L(v_i, G)$, of vertex v_i in G is then defined by

$$C_L(v_i, G) = \frac{(\Delta E)_i}{E_L(G)} = \frac{E_L(G) - E_L(G_i)}{E_L(G)}$$

[ea11]

To examine the intuition behind this formula, firstly, notice how the value of the fraction is between 0 and 1. This is because Laplacian energy is always positive (since we sum squares of eigenvalues) and $E_L(G_i) \leq E_L(G)$, since removing edges decreases Laplacian energy by Equation (2). Additionally, a vertex that is more well-connected will yield a higher centrality value, since the removal of that vertex causes a greater drop in energy (more edges have to be removed).

In the next section, we will apply our methods of analyzing graph connectivity to a real-world example: a collaboration network of jazz musicians. Our analysis will include a discussion of the Laplacian centralities of selected vertices. Readers interested in deriving Laplacian centralities in their own datasets and graphs may find scanning the following algorithm useful (taken from [ea18b]), which details the steps needed to calculate Laplacian centrality of a vertex in an unweighted graph (the source also provides algorithms for weighted graphs):

Algorithm 2 Dynamic Laplace Centrality Algorithm (unweighted version)

```
$V \leftarrow \{u_1, u_2, .., u_v\}$ , $E \leftarrow \{(i_1, j_1), (i_2, j_2), .., (i_e, j_e)\}$
$A_{dd} \leftarrow array\{(i_1, j_1), .., (i_n, j_n)\}$, $R_{emove} \leftarrow array\{(i_1, j_1), .., (i_n, j_n)\}$
procedure LapCentAddRemove($G \leftarrow (V, E)$, $A_{dd}$, $R_{emove}$, $\mathcal{C}_{entralities}$)
    $\mathcal{V}_s \leftarrow \{\}$
    for each $edge$ in $A_{dd}$ do
        $\mathcal{V}_s \leftarrow \mathcal{V}_s \cup edge.source() \cup edge.destination()$
        $G.add\_edge(edge)$
    end for
    for each $edge$ in $R_{emove}$ do
        $\mathcal{V}_s \leftarrow \mathcal{V}_s \cup edge.source() \cup edge.destination()$
    end for
     $\mathcal{V}_f \leftarrow \{\}$
    for each $node$ in $\mathcal{V}_s$ do
        $\mathcal{V}_f \leftarrow \mathcal{V}_f \cup G.neighbors(node)$
    end for
    for each $edge$ in $R_{emove}$ do
        $G.remove\_edge(edge)$
    end for
    $\mathcal{D}_{egrees} \leftarrow G.degrees()$
    for each $v$ in $\mathcal{V}_f$ do
        $\mathcal{N}_{eighbors} \leftarrow G.neighbors()$
        $loc \leftarrow \mathcal{D}_{egrees}[v]$
        $nei \leftarrow 2. \sum_{i=1}^{\mathcal{N}_{eighbors}} \mathcal{D}_{egrees}[i]$
        $\mathcal{C}_{entralities}[v] \leftarrow (loc^2 + loc + nei)$
    end for
    return $\mathcal{C}_{entralities}$, $|\ \mathcal{V}_f\ |$, $G$
end procedure
procedure Main
    $\mathcal{D}_{ataset} \leftarrow \{G_0, G_1, ..., G_n\}$, $A_{dd} \leftarrow \{A_0, A_1, ..., A_n\}$, $R_{emove} \leftarrow \{R_0, R_1, ..., R_n\}$
    $\mathcal{C}_{entralities}$, $\mathcal{N}_{um}\mathcal{C}_{entralities} \leftarrow$ LapCent($G_0$)        ▷ initial step in the full network
    $G \leftarrow G_0$, $i \leftarrow 1$
    while ($i \leq |\mathcal{D}_{ataset}|$) do        ▷ calculate centralities for the increments
        $\mathcal{C}_{entralities}$, $\mathcal{N}_{um}\mathcal{C}_{entralities}$, $G \leftarrow$ LapCentAddRemove($G$, $A_{dd}[i]$, $R_{emove}[i]$, $\mathcal{C}_{entralities}$)
    end while
end procedure
```

Laplacian centrality, though a less common centrality measure than others, turns out to be quite useful in several applications. For example, a possible extension of Laplacian centrality to directed graphs has been suggested by [Nan17] to analyse diffusive systems, such as heat dissipation and the spread of viruses.

7 Analyzing the Jazz Musician Network

Up to this point, we have developed several tools for network analysis and applied them to graphs with relatively few vertices and edges. This made the graphs and the theory easier to visualize. Where the tools become most powerful, though, is when they are applied to the larger, more complex networks found in the real world.

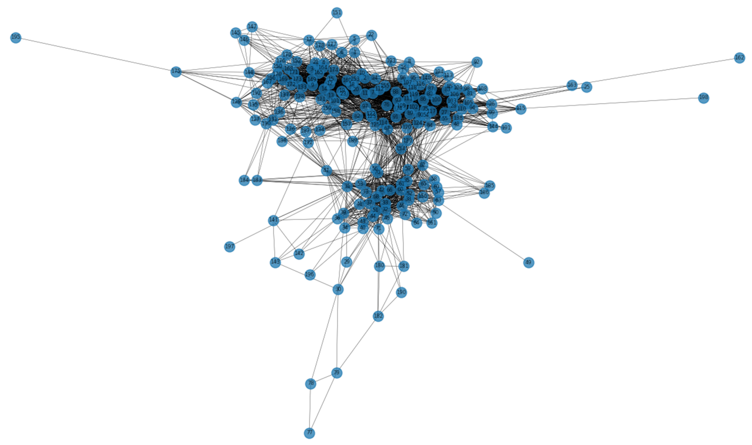
In this section, we will use our linear algebra tools to analyze a collaboration network of jazz musicians, with 198 nodes and 2,742 edges. The data was collected in 2003 and is available for download on [kon17]. We will answer questions like, is the network connected, and if so, how connected? What can we say about the network's underlying structure? Ultimately, what can we learn about collaboration in jazz?

In the network, each node is a jazz musician and each edge represents collaboration; specifically, an edge between Musician 1 and Musician 4, for example, denotes that Musicians 1 and 4 have played together in a band.

In the Python snippet below, we have downloaded the dataset into a file called `jazz_musicians.txt`, which lists every edge in the jazz collaboration network. We then use functions in the `networkx` and `matplotlib` packages to construct and visualize the graph corresponding to the network:

```
import matplotlib.pyplot as plt
import networkx as nx
# We will need the numpy package later
import numpy as np

# Create a graph G using the list of edges in the jazz_musicians.txt file
G = nx.read_edgelist("jazz_musicians.txt")

# Draw and customize G ('width' specifies edge width and 'alpha' specifies transparency of nodes)
nx.draw_networkx(G, node_size=150, font_size=6, width=0.4, alpha=0.75)

# Display G
plt.show()
```

Our graph G looks like this:



We can immediately see that there are a few nodes, like the ones labelled 49 and 195, with degree 1. These musicians have only collaborated with one other musician. We also see very dense "hubs" of nodes, suggesting very active communities of musicians collaborating with one another.

We can also get the adjacency and Laplacian matrices of G:

```
# Construct the adjacency and Laplacian matrices of G
adj = nx.adjacency_matrix(G)
lp = nx.laplacian_matrix(G)
```

Now, let us take a look at the Laplacian spectrum:

```python
# Get and print the Laplacian spectrum
laplacian_spectrum = nx.laplacian_spectrum(G)
print("The eigenvalues of L(G) are: " + np.array_str(laplacian_spectrum, suppress_small=True))
```

Here is an abbreviated output:

The eigenvalues of $L(G)$ are: $[0, 0.5719939, \ldots, 97.04242603, 101.18320277]$

The multiplicity of eigenvalue 0 is one, so our jazz network is connected! There is a path between every pair of musicians. We can also see that the algebraic connectivity of the network is 0.5719939. This is the Fiedler Value of G. Knowing this value, we can study how the connectivity of the network can be strengthened. We might answer questions like, how can we create a tighter-knit jazz community? What would this community look like? See [ea18a] for one connectivity-maximizing algorithm.

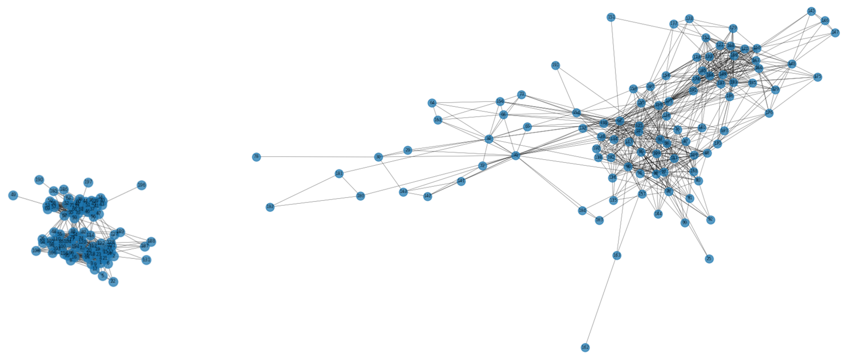
Let us try partitioning our graph. Computing our Fiedler Vector, we see that there are far more negative entries than positive ones. Then, our standard approach of clustering nodes by the sign of the corresponding entry in the Fiedler Vector would not be very satisfactory. To address this issue, we will follow [WG21] in adapting the Fiedler Method. We will put the nodes corresponding to the smaller half of the entries in the Fiedler Vector in one partition and the rest in a second partition. This might require cutting more edges, but it will give equally sized subgraphs. Here is the code that accomplishes this:

```python
# Get the Fiedler Vector of G
fiedler_vector = nx.fiedler_vector(G)

# Declare lists to store the nodes in each partition
partition_one = []
partition_two = []

# Partition the graph using the Fiedler Vector according to our 'half and half' approach
for i in range(nx.number_of_nodes(G)):
    if fiedler_vector[i] < np.median(fiedler_vector):
        partition_one.append(str(i + 1))
    else:
        partition_two.append(str(i + 1))
```

We now construct and draw our subgraphs:

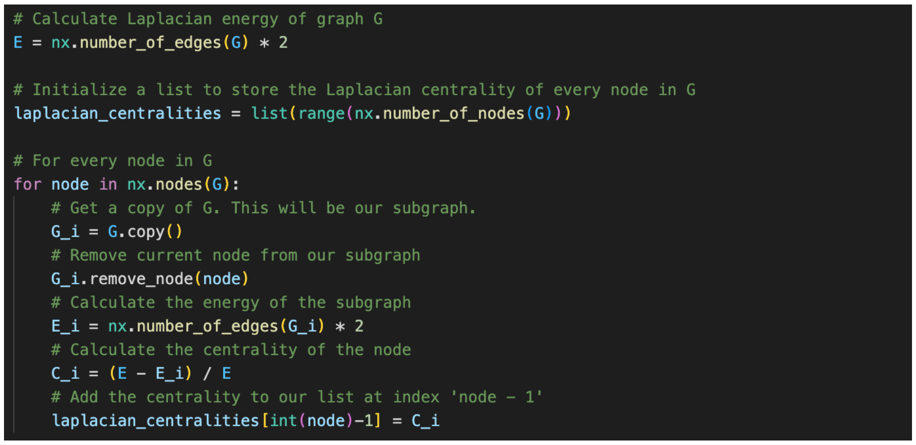
```python
# Construct and display the subgraph corresponding to our first partition of G
subgraph_one = nx.subgraph(G, partition_one)
nx.draw_networkx(subgraph_one, node_size=150, font_size=6, width=0.4, alpha=0.75)
plt.show()
```

We repeat the same to get a second subgraph from `partition_two`. We end up with `subgraph_one` shown below on the left and `subgraph_two` on the right, with 99 nodes in each:



What can we infer about the jazz musician network from our partitioning? We observe that subgraph_one is far more dense than subgraph_two, which is much more spread out visually. In fact, we find that the Fiedler Value is about 0.8 for subgraph_one and about 0.5 for subgraph_two. The musicians in subgraph_one are more connected; they have collaborated with each other much more than the musicians in subgraph_two. We might infer that our partitioning hints at an underlying structure in the jazz musician network; perhaps subgraph_one is the 'heart' of the network, representing a community of older, more accomplished musicians who are more collaborative or have had more time to collaborate, while subgraph_two represents another community of younger, up-and-coming musicians who are still making new connections in the music industry.

Which jazz musicians are most central to the collaboration network? Which are least central? Let us calculate the Laplacian centrality of every node using the following code:



```
# Calculate Laplacian energy of graph G
E = nx.number_of_edges(G) * 2

# Initialize a list to store the Laplacian centrality of every node in G
laplacian_centralities = list(range(nx.number_of_nodes(G)))

# For every node in G
for node in nx.nodes(G):
    # Get a copy of G. This will be our subgraph.
    G_i = G.copy()
    # Remove current node from our subgraph
    G_i.remove_node(node)
    # Calculate the energy of the subgraph
    E_i = nx.number_of_edges(G_i) * 2
    # Calculate the centrality of the node
    C_i = (E - E_i) / E
    # Add the centrality to our list at index 'node - 1'
    laplacian_centralities[int(node)-1] = C_i
```

Examining the resulting list of centralities (code not shown), we find that vertex 67 has the highest Laplacian centrality in the network, with a centrality of about 0.0365. We might conclude that removing the musician corresponding to this vertex would then have the biggest (negative) impact on collaboration in jazz. Vertices 49, 162, 195, 197, and 198 are the least central, all with centralities of about 0.000365. Vertex 5 is somewhere in between, with a centrality of about 0.0044. These values make sense: vertex 67 has degree 100, vertex 5 has degree 12, and vertices 49, 162, etc. have degree 1. Removing vertex 49, for example, would then represent the loss of just one collaborative link, causing a low drop in Laplacian energy.

Observe how all the centralities in this network are relatively small (less than 0.1). No one musician is vastly more important than the others; there is no single node we could remove that would significantly hurt the network's energy as a whole. This suggests a certain robustness in the jazz collaboration network. The contributions to collaboration are relatively spread out among the musicians. We saw that the most 'central' musician is linked to 100 other musicians; but considering the network has 2,742 total edges, this represents just one relatively small, yet very valuable, contribution to the larger jazz community.

8 Conclusion

The connection between linear algebra and graph theory is fascinating and as we have seen, provides us with powerful methods for gaining insight into complex networks. Our paper is only an exposition to this intersection, and we have used considerable space introducing preparatory definitions, terminology, and examples; nonetheless, we have seen that with preliminary tools, we can gain insight into the connectivity of a network of 198 jazz musicians and interpret this insight by considering the possible community structure underlying the network as well as the individual influence of particular musicians. The possibilities of what can be explored with more advanced tools and more complicated networks involving directed and/or weighted graphs are then very exciting.

References

[Bap14] Ravindra B. Bapat. Graphs and Matrices. https://doi.org/10.1007/978-1-4471-6569-9_4, 2014.

[Dai14] Madelaine Daianu. Algebraic connectivity of brain networks shows patterns of segregation leading to reduced network robustness in alzheimer's disease. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4669194/, 2014.

[ea11] Xingqin Qi et al. Laplacian centrality: A new centrality measure for weighted networks. http://www.math.wvu.edu/~cqzhang/Publication-files/my-paper/INS-2012-Laplacian-W.pdf, 2011.

[ea18a] Gang Li et al. Maximizing Algebraic Connectivity via Minimum Degree and Maximum Distance. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8412478, 2018.

[ea18b] Mario Cordeiro et al. Dynamic laplace: Efficient centrality measure for weighted or unweighted evolving networks. https://arxiv.org/pdf/1808.02960.pdf, 2018.

[IG05] Bo Zhou Ivan Gutmann. Laplacian energy of a graph. https://www.aimath.org/WWN/matrixspectrum/gutman2006.pdf, 2005.

[Joh17] Dylan Johnson. Graph Theory and Linear Algebra. https://www.math.utah.edu/~gustafso/s2017/2270/projects-2017/dylanJohnson/Dylan%20Johnson%20Graph%20Theory%20and%20Linear%20Algebra.pdf, 2017.

[kon17] Jazz Musicians Network Dataset – KONECT. http://konect.cc/networks/arenas-jazz, 2017.

[Kotnd] Pravesh Kothari. Spectral Graph Theory. https://www.cs.cmu.edu/~venkatg/teaching/15252-sp20/notes/Spectral-graph-theory.pdf, (n.d.).

[Mar13] Anne Marsden. Eigenvalues of the Laplacian and Their Relationship to the Connectedness of a Graph. https://math.uchicago.edu/~may/REU2013/REUPapers/Marsden.pdf, 2013.

[Nan17] Alice Nanyanzi. Laplacian matrix and applications. https://www.math-berlin.de/images/summerschool/randgraph-2017/Nanyanzi-Alice_presentation.pdf, 2017.

[Tru94] Richard J. Trudeau. Introduction to graph theory. https://www.pdfdrive.com/introduction-to-graph-theory-e157488969.html, 1994.

[vdH13] Martijn P. van den Heuvel. Network hubs in the human brain. https://www.cell.com/trends/cognitive-sciences/fulltext/S1364-6613(13)00216-7?_returnURL=https%3A%2F%2Flinkinghub.elsevier.com%2Fretrieve%2Fpii%2FS1364661313002167%3Fshowall%3Dtrue, 2013.

[Wal19] Julia Walchessen. Eigenvalues of the Laplacian on a Graph. https://math.uchicago.edu/~may/REU2019/REUPapers/Walchessen.pdf, 2019.

[WG21] Justin Wyss-Gallifent. Graph theory. https://www.math.umd.edu/~immortal/MATH401/book/ch_graph_theory.pdf, 2021.