June 8, 2021
CMPSC 292F: Graph Laplacians

Project Report
Evaluation and Approximation of Graph Connectivity Metrics

Ashley Bruce and Jacqueline Mai

ABSTRACT

The conductance of a graph, also known as Cheeger's constant, is a measure of how "well-knit" the graph is. The Fiedler value, the second smallest eigenvalue of a graph, gives us a measure of the algebraic connectivity among the elements in a network. The isoperimetric number is another metric of connectivity similar to the conductance that measures volume by quantity of vertices rather than total degree of a cut set. Our project is a case study that looks at the comparison of these values across graphs of different sizes and of differing levels of connectivity. We determine the calculation limitations and look at how accurate various approximations of these metrics are.

INTRODUCTION

There are a number of measures of connectivity for a graph; for example, the isoperimetric number indicates the narrowest bottleneck, and the conductance indicates how easy it is to completely isolate some set of vertices from the rest of the graph. Calculating either of these metrics is an NP-hard problem, since they both require enumerating every unique cut in the graph G. There are 2^{n-1} unique cuts in a graph with n vertices, which means calculating the conductance or isoperimetric number directly requires exponential time in proportion to the size of the graph.

Since we are often interested in evaluating the connectedness of a given graph, work has been done to approximate both the isoperimetric number and the conductance. Specifically, the Cheeger inequalities offer ways of determining bounds for those metrics using the Fiedler values of the graph Laplacian and normalized Laplacian matrices, respectively.

In our project, we perform a series of experiments that demonstrate the limitations of calculating the conductance and isoperimetric number directly for a number of graphs. We then calculate the Fiedler values of the Laplacian and normalized Laplacian and verify the Cheeger inequalities experimentally. We apply the algorithms to a series of increasingly more connected graphs and analyze how accurately the bounds reflect the trend of connectedness. Finally, we compare the runtimes of all of our

experiments to show that, while the Cheeger inequalities don't provide a tight bound, they provide a reasonable approximation at a much lower complexity.

METHODS

The Fiedler value is defined as the second smallest eigenvalue of the Laplacian of a graph.
The conductance of a graph is formally defined as follow [1]:

$$\phi(S) = \frac{|\partial S|}{\min(d(S), d(V - S))}$$

The numerator of the equation looks at the edges that lie in the interface between the sets S and V-S. The denominator takes the minimum between the degrees of sets S and V-S.

The isoperimetric ratio of a graph is formally defined as follows [2]:

$$\theta(G) = \min_S \frac{|\partial(S)|}{\min(|S|, |V - S|)}$$

The numerator of this equation looks at the edges that lie in the interface between the sets S and V-S. The denominator only considers set sizes with at most half the vertices of the graph.

```
168     size = len(A)
169     numPartitions = 2 ** (size - 1)
170     for i in range(1, numPartitions):
171         binArray = np.zeros(size)
172         binString = str(bin(i))
173         binNums = binString[2:]
174         counter = len(binArray) - 1
175         for digit in reversed(binNums):
176             binArray[counter] = int(digit)
177             counter -= 1
178
179         # SET PARTITIONING
180         ones = []
181         zeros = []
182         for j in range(len(binArray)):
183             if binArray[j] == 1:
184                 ones.append(j)
185             else:
186                 zeros.append(j)
```

In order to determine what both the conductance and the isoperimetric ratio of the overall graph are, all the nodes of the graph need to be partitioned into two sets in every possible way, where the set they are partitioned into does not matter.

To partition the nodes into two unique sets, we decided to use properties of the binary representation of the size of the array, following the algorithm outlined in [6]. Binary representations of the number will give us a unique ordering that we can use to split up our items into one set or

the other based on the 1s and 0s in our binary value.

Since we are only looking at undirected graphs, we only have to go up to 2^{n-1}, as the graph is symmetric and we do not have to double check the partitioning. This halves the time when looking at larger matrices. We could amend the code to check all combinations over both sets by changing (size - 1) to just size, but have elected to keep it this way for the sake of optimization.

To better visualize this, consider the following set A = {1, 2, 3, 4}.
Partitioning the items into the following sets: Set1 = {1}, Set2 = {2, 3, 4} would be the same as:
Set1: {2, 3, 4}, Set2 = {1}, because which set the item was partitioned in does not matter. If the graph was directed, it would matter which set we partitioned them into and Set1 and Set2 would implicate different things.

In the code shown here, we determine the number of unique partitions there are for our set (line 169) and use that number to loop through the rest of our code, as we need to calculate the conductance every iteration and find the minimum conductance. We take the number iteration we are on, and convert it to a binary number (lines 172-173). Since we need the leading 0s of our binary number for proper set partitioning, lines 174-177 add the leading 0s into an array so the length of each binary number corresponds to the number of elements present in the graph.

```
188         capacity = 0
189         for one in ones:
190             for zero in zeros:
191                 if A[one, zero] != 0:
192                     capacity += A[one, zero]
193
194         L = csgraph.laplacian(A)
195         minEdges = setDegrees(L, ones, zeros)
196         if len(ones) < len(zeros):
197             capacity = capacity / minEdges
198         else:
199             capacity = capacity / minEdges
200
201         if capacity < conductance:
202             conductance = capacity
```

The actual set partitioning step can be seen on lines 180-186. We separate the items based on their binary representation of 1 or 0 into the sets and then use this to check whether or not there are edges in the intersection between the two sets.

To do this, we iterate through all the items in the first set and compare them to every item in the second set (lines 189-190), and check to see if there is an edge between them (line 191). If there is, we add it to the running total for capacity (line 192).

We then determine which set contains the smaller number of edges and use that to calculate the capacity for the current iteration (lines 194-199). If the capacity of the current iteration is smaller than the current minimum conductance, we update the minimum conductance, and at the end will have the conductance for the graph.

Calculating the isoperimetric number was essentially the exact same process, but we just divided the capacity by the size of the smaller set in the cut instead on lines 196-199. By selecting the smaller set every time over the set of unique cuts, we ensure that we only ever use sets that are at most $n/2$ vertices in our calculations.

Calculating the Fiedler value was more straightforward. We were able to utilize methods provided by the scipy library and obtain all the eigenvalues using that. To get the Fiedler value, we sorted all the eigenvalues and took the second value.

Approximating the conductance of a graph requires obtaining the Fiedler value from the normalized Laplacian of the graph. There are three ways to do this, but they all require the same matrix, $D^{-½}$. The matrix D is the matrix with the degree of each node on the diagonal, and 0s everywhere else. To get $D^{-½}$ from D, you perform the operation shown to the right using the entries from matrix D.

$$D^{-1/2} = \begin{pmatrix} \frac{1}{\sqrt{d(1)}} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sqrt{d(2)}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sqrt{d(n)}} \end{pmatrix}$$

The first way to get the normalized Laplacian is to first calculate the normalized adjacency matrix, then subtract that from the identity matrix.

$$\mathscr{A} \equiv D^{-1/2} A D^{-1/2} \qquad \mathscr{L} \equiv I - \mathscr{A}$$

The next two ways are similar. You can multiply $D^{-½}$ on each side of the Laplacian, or on the Diagonal Matrix, D, minus the original adjacency matrix.

$$D^{-1/2} L_G D^{-1/2} \qquad D^{-1/2}(D - A)D^{-1/2}$$

All of these means of calculating the normalized Laplacian are equivalent [3]. With this normalized Laplacian, we then took the Fiedler value and found an upper and lower bound for the conductance. This will be touched upon further in a later section of this paper.

We then ran our code on various sparse graphs. As mentioned earlier, we required our matrices to be symmetric to cut down on the running time of our algorithm. This limited our graphs to undirected graphs of small sizes. We obtained some graphs from sparse graph websites [4], but as the sizes were limited, we decided to create our own graphs of various sizes as well. This was in part due to some of the newer mat files giving us issues when trying to import them into Python, so we had to stick to older matrices, limiting our selection even further.

The graphs we ran our tests on can be seen in the following two figures. The graphs obtained from [4] included an Adjacency matrix we could extract as a numpy array. For the graphs we created, we wrote out the adjacency matrices using lists and converted them to numpy arrays for calculations. We also decided to leave them unweighted (essentially just giving every edge a weight of 1) for simplicity.

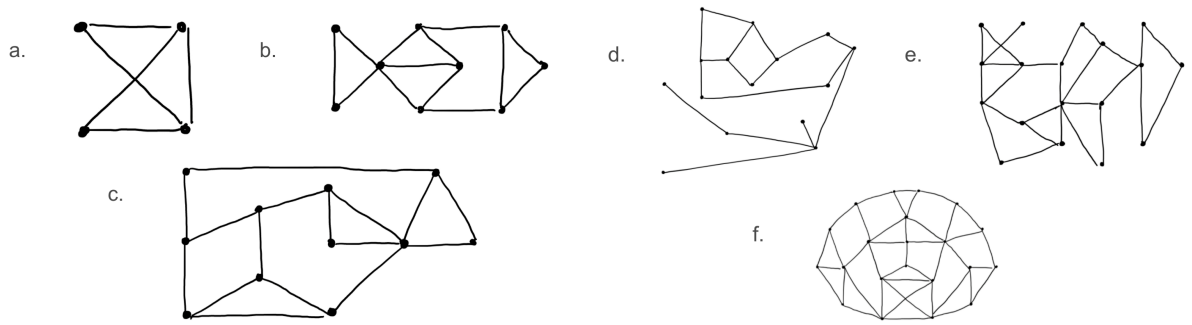

Figure 1. a) ours_4, b) ours_9, c) ours_11, d) ours_15, e) ours_18, f) ours_21

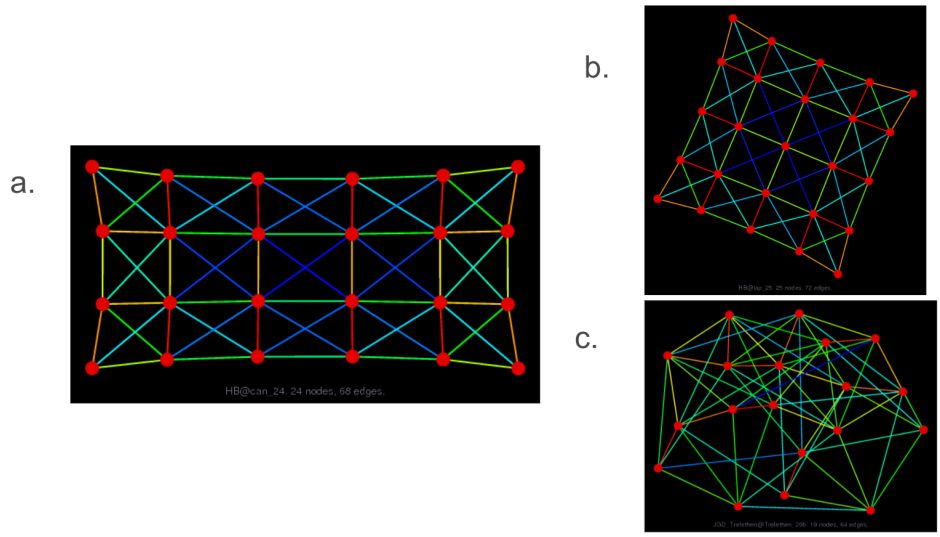

Figure 2. Graphs obtained from [4]. a) can_24, b) lap_25, c) Trefthen_20b

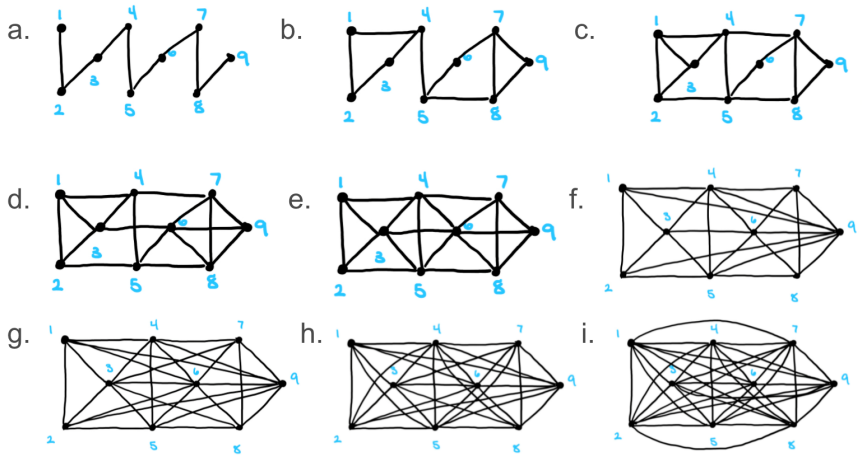

Figure 3. A graph of 9 nodes with differing levels of connectivity. a) nine_1, b) nine_2, c) nine_3, d) nine_4, e) nine_5, f) nine_6, g) nine_7, h) nine_8, i) nine_9. The graph seen in (a) is minimally connected and the graph seen in (i) is maximally connected.

RESULTS

We first looked at how the conductivity of a graph changed with the addition of edges. To do this, we started off with a graph with 9 nodes and minimally connected the graph. The conductance and Fiedler value were then calculated. A few edges were added to the original graph and the conductance and Fiedler value were calculated again. We repeated this process with more edges, updating the graph, until we had a complete graph. These values are summarized in the table below.

Graph	Number of Edges	Conductance	Isoperimetric Ratio	Fiedler
nine_1	8	0.1429	0.25	0.1206
nine_2	11	0.1111	0.25	0.2505
nine_3	14	0.2308	0.75	0.7349
nine_4	17	0.25	1	1.1535
nine_5	19	0.2941	1.25	1.2954
nine_6	23	0.4286	2.25	2.4778
nine_7	27	0.52	3.25	4
nine_8	31	0.5556	3.75	5
nine_9	36	0.625	5	9

Table 1. Comparison of conductance, isoperimetric ratio, and the Fiedler value as more edges are added. nine_1 is a minimally spanning graph and nine_9 is a complete graph.

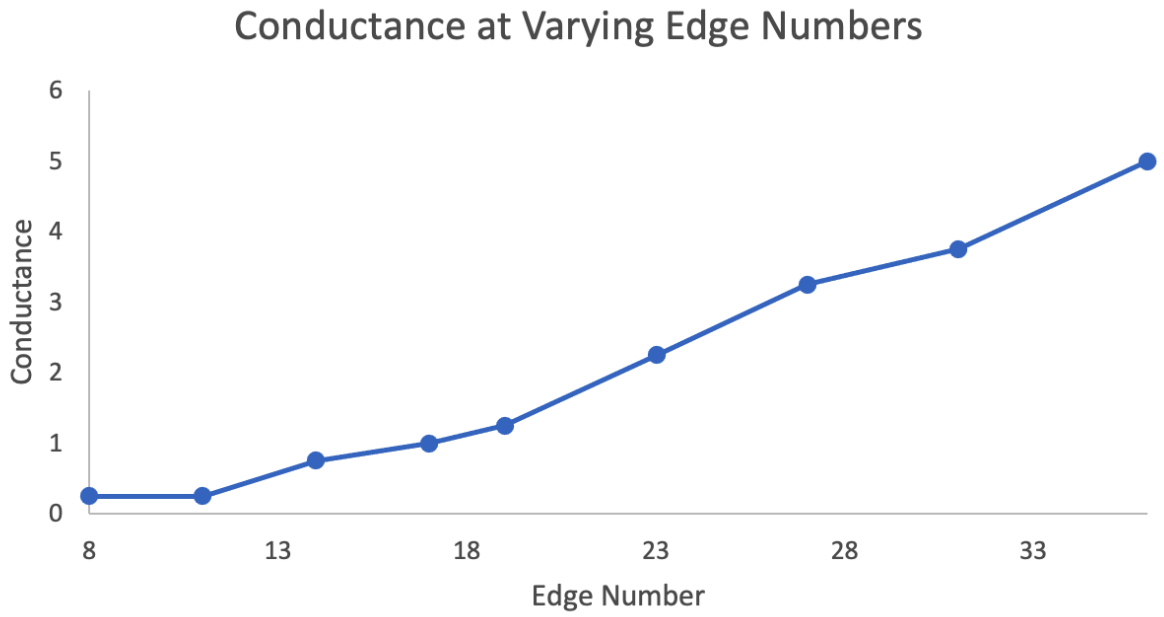

Figure 4. Plot of the data from Table 1. Conductance increases as edge number increases when adding edges to the same graph. The size of the graph was 9.

The results from this experiment confirmed what we had expected. The more connected the graph is, the higher the number both the conductance and the Fiedler value is. Another notable result is that the Fiedler value of the complete graph (nine_9) was 9, which corresponds to the size of the graph. This is due to the characteristics of the eigenvalues of a complete graph. A complete graph will have an eigenvalue of n, n-1 times, with the last eigenvalue being 0.

We then looked at the conductivity of graphs of different sizes in comparison to their Fiedler value. These results can be seen in Table 2.

When initially defining our functions, we had created two separate functions for calculating the conductance: one for the graphs we created ourselves and one for the graphs we downloaded from a mat file. The functions were the same except for where we indexed the array to determine whether or not there was an edge between the two nodes. We did this because the array given in the mat file was a numpy array, while the array we created ourselves was a normal array. When doing it this way initially, we discovered that there was significantly more overhead when having to use numpy arrays, as the 21x21 matrix we created took less time to calculate conductance than did the 19x19 matrix we downloaded. To fix this, we cast all our arrays to numpy arrays and used the same function for all our conductance calls. This allowed us to more accurately compare time measures for each graph.

Graph	Size (nxn)	Conductance	Time	Isoperimetric Ratio	Time	Fiedler
ours_4	4x4	0.6	<1s	1.5	<1s	2
ours_9	9x9	0.25	<1s	0.666	<1s	0.5459
ours_11	11x11	0.2	<1s	0.6	<1s	0.7628
ours_15	15x15	0.11	1s	0.2	<1s	0.1331
ours_18	18x18	0.134	7s	0.375	6s	0.2085
Trefethen_20b	19x19	0.3	34s	2	31s	2.9499
ours_21	21x21	0.1892	1 min 9s	0.7	53s	0.5440
can_24	24x24	0.1471	26 min 35s	0.833	22 min 18s	0.6654
lap_25	25x25	0.2059	56 min 9s	1.167	52 min 3s	0.9801

Table 2. Initial results obtained by calculating the conductance and Fiedler value on the specified graphs.

We made note of how long it took to calculate the conductance for every graph. It wasn't until a graph size of 18x18 where the computations were no longer instantaneous. As calculating the conductance is an exponential-time algorithm, we can see the time it takes to finish calculating quickly increases with only a minor number of node additions. At only 25 nodes, calculating the conductance took almost an hour, so we decided not to increase the graph size any more.

APPROXIMATING CONDUCTANCE

As we mentioned before, the challenge with computing conductance is that there is no way to avoid enumerating every possible unique cut. No matter how much we optimize the specific operations in our computation, we still need to iterate 2^{n-1} times for an $n*n$ matrix. For this reason, similar to many other NP-hard algorithms, work has been done to find a way to approximate the conductance within a reasonable time. After finding the limitations of our direct approach, we read [5], which offered the following theorem:

Theorem 3.27 (Cheeger's inequality). *For a graph G,*

$$\frac{\lambda_2}{2} \leq \phi(G) \leq \sqrt{2\lambda_2} \tag{3.28}$$

where λ_2 is the second smallest eigenvalue to the normalized Laplacian.

This inequality actually followed with our initial intuition pretty well- we had looked for a relationship between the Fiedler value of a graph and its conductance, and this theorem offers a relationship between the Fiedler value of the graph's normalized Laplacian and its conductance. To confirm this experimentally, we wrote a function to calculate a graph's normalized Laplacian and used the same code we'd used earlier to get the Fiedler value, then used that value to determine how accurate the bounds of the theorem are to the conductances we had directly calculated earlier. The time it took to calculate the approximation was almost instantaneous, so the times are not included in this table.

Graph	Lower Bound	Conductance	Upper Bound
ours_4	0.5	0.6	1.4142
ours_9	0.1120	0.25	0.6693
ours_11	0.1346	0.2	0.7338
ours_15	0.0339	0.11	0.3684
ours_18	0.0428	0.134	0.4139
Trefethen_20b	0.2323	0.3	0.9640
ours_21	0.0797	0.1892	0.5645
can_24	0.0671	0.1471	0.5182
lap_25	0.0999	0.2059	0.6322

Table 3. Approximation bounds for conductance compared with the actual conductance. Approximated values are based on the Fiedler value of the normalized Laplacian.

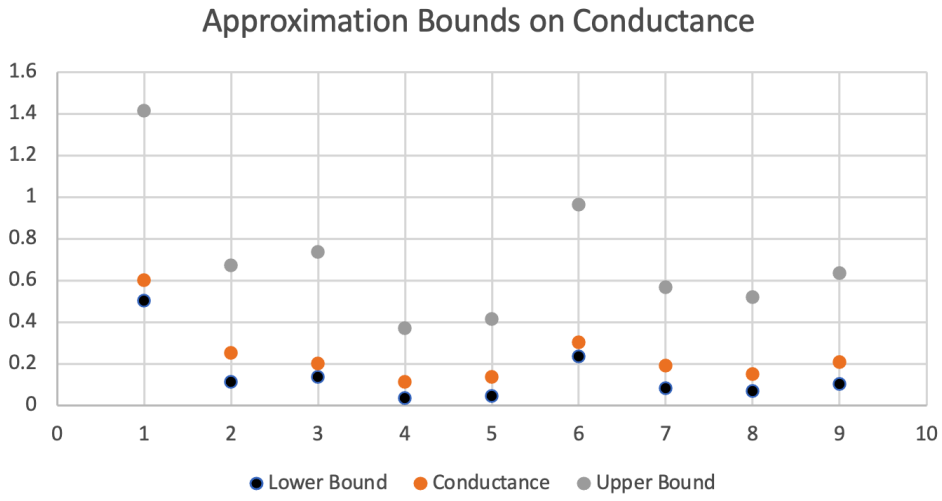

Figure 5. Plot of the data from Table 3. Conductance values fall between those approximated from the upper and lower bounds.

As can be seen in the table, the conductance of the graph falls between both bounds for all values. Because these values can be calculated almost instantaneously, this can be a good approximation when dealing with larger sized graphs. This range is pretty broad though, so if a more exact value for conductance is needed, it would probably be best to either take the time to calculate the conductance, or find an approximation with a tighter bound. It appears though that in the graphs we have chosen, the actual conductance tends to be closer to that of the lower bound. While this may be entirely coincidental, it is still an interesting observation.

APPROXIMATING THE ISOPERIMETRIC NUMBER

As we mentioned before, we can also approximate the isoperimetric number with the Fiedler value of the Laplacian of the graph. The isoperimetric theorem, another one of the Cheeger inequalities, defined in [1] gives us:

$$\theta_G \geq \lambda_2/2.$$

We can easily compute this bound for all of our graphs from the Fiedlers presented in Table 2:

Graph	Lower Bound	Isoperimetric Ratio
ours_4	1	1.5
ours_9	0.2730	0.666
ours_11	0.3814	0.6
ours_15	0.0665	0.2
ours_18	0.1043	0.375
Trefethen_20b	1.4750	2
ours_21	0.2720	0.7
can_24	0.3327	0.833
lap_25	0.4901	1.167

Table 4: Lower bound approximation of the isoperimetric number using the Fiedler value, compared with the actual isoperimetric ratio of each graph.

As we can see from Table 4, the inequality does hold, so we can always easily get a lower bound on the isoperimetric ratio of any graph. This means the Fiedler value of a graph gives us a decent idea of the

surface-to-volume ratio of a graph, but not to a tight bound by any means. There are a few cases in Table 4 where the lower bound approximation actually comes pretty close to the actual value, but for the most part it is significantly smaller than the true value of the isoperimetric ratio. This inequality also doesn't give us an upper bound, which makes it even harder to determine how tight the lower bound could actually be.

FINAL REMARKS

In this project, we looked at how the conductance, isoperimetric ratio, and Fiedler value changed over graphs of different sizes and differing levels of connectivity. It was shown how the more connected a graph is, the higher all three values will be. We have also shown how quickly the time limitations of calculating both the conductance and the isoperimetric ratio of a graph. While trying to find an exact measure of these is NP-hard, there are approximations that give good bounds for these values in a small fraction of the time. It was shown that these bounds held for all the graphs tested.

If we were to extend this project, we could try testing the bounds of the performance of our approximation algorithms. We could pass increasingly large graphs into our functions and see at what point even the approximations start to take an unreasonable amount of time to complete. It might also be interesting to try optimizing the code itself. As this project showed, we know that our mathematical operations are all correct, but our code was a very straightforward implementation of the formulae. We might try using more clever data structures or strategies to cut down on performance or memory overhead.

WORKS CITED

[1]. https://github.com/johnrgilbert/S21-graph-laplacians/blob/main/Readings/index-latest.pdf
[2]. Private lecture notes owned by Ambuj Singh at UCSB for 292F, Graph Machine Learning
[3]. https://people.orie.cornell.edu/dpw/orie6334/Fall2016/lecture7.pdf
[4]. https://sparse.tamu.edu/
[5]. http://math.uchicago.edu/~may/REU2020/REUPapers/Zhang,Yueheng.pdf
[6].https://stackoverflow.com/questions/6999460/algorithm-for-all-possible-ways-of-splitting-a-set-of-elements-into-two-sets

APPENDIX

Public Github: https://github.com/ashleyybrucee/GraphLaplaciansConductance