1 Introduction

When designing algorithms, there are certain properties of the algorithm that we aim to optimize for. These include:

- Time
- Space/Memory
- Parallelism
- Power
- Loss/Accuracy
- Determinism / Randomness
- Communication

The main goal of this class will be focusing on the Space / Memory, Determinism / Randomness, and Communication complexity of algorithms.

1.1 Memory Efficiency

We define the memory usage of an algorithm based off how much *extra memory* it requires given an input of size x. We say that the algorithm as read access to the input x and Read/Write access to any extra space it may need to run.

2 s-t Connectivity

The s-t connectivity problem is defined as follows:

- Input: $G = (V, E), s, t$ where s is the source node and t is the destination node.
- Output: $\begin{cases} \text{YES if } s, t \text{ are connected.} \\ \text{NO otherwise.} \end{cases}$

The trivial solution to this problem is to perform a BFS or DFS from s to see if t is reachable. This requires a graph representation which is typically assumed to be one of the following:

- **Adjacency List**: Each vertex has a corresponding list of vertices that it shares an edge with.

- **Adjacency Matrix**: An $(|V| = n)^2$ sized matrix denoted as A where $A[i, j] = 1$ if there is a directed edge from vertex i to vertex j.

2.1 Basic algorithm

The basic algorithm using BFS is as follows

Algorithm 1 Graph Reachability

```
1: Flag[u] ← 0   ∀u
2: Flag[s] ← 0
3: Q ← [s]
4: while Q ≠ ∅ do
5:     remove v from Q
6:     for each neighbor u of v do
7:         if Flag[u] = 0 then
8:             add u to Q
9:         end if
10:    end for
11:    Flag[v] ← 1
12: end while
13: if Flag[t] = 1 then
14:    output YES
15: else
16:    output NO
17: end if
```

The running time of this algorithm is $O(|V| + |E|)$. The Memory usage is dependent on the Flag array which requires $|V|$ extra bits. (Ask why this is the bound since the algorithm needs $\log(|V|)$ bits to represent nodes).

2.2 Memory Efficient s-t Connectivity

Questions were raised wondering if there was a more memory efficient version of BFS that solved this problem and if so, what is the least memory needed to solve s-t connectivity.

Theorem 2.1 (1980s)**.** *There is a randomized algorithm with* $5 \log |v|$ *bits of additional memory to solve* $s - t$ *connectivity problem that runs in* $O(n^3)$ *runtime.*

Random Walk Algorithm

- Counter $\leftarrow 0$
- $v \leftarrow s$
- **while** Counter $\leq T$:
  - **if** $v = t$ **then return YES**
  - **else**
    - $v \leftarrow$ a random neighbor of v
    - Counter $\leftarrow$ Counter $+ 1$
- **return NO**

The bits needed are for current vertex $v(\log n)$, Counter $(\log T)$,
Bits needed for random sampling $(\log n)$.
The above algorithm uses the following result:

Theorem 2.2. *For all graphs, if s, t are connected:*

$$\mathbb{E}[\textit{Time random walk from } s \textit{ hits } t\,] = O(n^3)$$

2.3 Notes and History

The above algorithm and proof led to the question of whether or not we can get a **logspace** algorithm for directed s.t. connectivity **without randomness**?

The reason that this question holds significance in complexity theory is its implications on computational ability. s-t Connectivity is a problem that is NL (Nondeterministic Logarithmic Space) Complete. This has the following implication:

Theorem 2.3. *If directed $s - t$ connectivity can be solved with $O(\log |V|)$ bits of memory (and no randomness) $\Longrightarrow$ Any randomized algorithm can be made deterministic at the expense of a constant-factor increase in memory (and polynomial slow-up in time).*

In 2005, Omer Reingold proved that $s.t$ connectivity on undirected graphs can be solved with $O(\log |v|)$ extra memory. This is the first big hit for this class.

3 Spectral Graph Theory

To construct an algorithm that solves $s.t$ connectivity, we use an adjaceny matrix. We also need the following definitions.

Definition 3.1. We say a graph is $D-$regular if all the vertices have the same degree D.

Definition 3.2. A Normalized Adjacency Graph, denoted as $M_a = \frac{A_G}{D}$. This means the sums of the matrix add to 1.

(I don't really know how these relate, ask professor next lecture)

Theorem 3.1 (SGT Theorem 1)**.** *If G is regular. Then:*

1. *1 is an eigenvalue of M_G*
2. *G is connected $\iff$ eigenvalue of 1 is unique*

Proof. (1) To demonstrate the first part of the theorem, we simply need to show a vector that, when multiplied by M_G, remains the same. Consider the $\vec{1}$ (All 1's vector). Since the matrix is normalized, the sum of all entries of a particular row is

$$\sum_{i=1}^{d} \frac{1}{d} = 1$$

Thus, $\vec{1}$ multiplied by this matrix must result in a 1 for every row which gives us back the 1 vector □

We prove the second part of the theorem in two parts.

1. All eigenvalues are less than or equal to 1
2. If the graph is disconnected, the eigenvalue 1 appears at least twice in the eigenvalues of the matrix

Proof. (2a) To demonstrate that all eigenvalues are less than or equal to 1, consider v_{i*}, the largest entry in v in absolute value. We will show that this

value cannot increase as a result of the matrix multiplication.

$$\begin{aligned}
\lambda \cdot v_{i*} &= \sum_{j=1}^{n} M[i,j] \cdot v_j \\
|\lambda| \cdot |v_{i*}| &= |\sum_{j=1}^{n} M[i,j] \cdot v_j| \\
&\leq \sum_{j=1}^{n} M[i,j] \cdot |v_j| \\
&\leq \sum_{j=1}^{n} M[i,j] \cdot |v_{i*}| \\
&= |v_{i*}| \cdot \sum_{j=1}^{n} M[i,j] \\
&= |v_{i*}| \cdot 1
\end{aligned}$$

Here we see that based on the normalized rows, the max entry in the vector cannot increase in magnitude. This implies that the absolute value of λ must be less than or equal to 1. □

To prove that a disconnected graph has at least 2 eigenvectors with eigenvalue 1, consider the structure of a matrix that represents a disconnected graph. This would be a **block structure** meaning that it would contain two squares in the matrix that correspond to the connected components of the graph. In this case, the vectors of only 1's for those particular entries would result in the same vector when multiplied by that matrix. (Look at notes for this better intuition).

3.1 Finalizing Proof (Lecture 2 Start)

Claim: G is connected and has self-loops $\Rightarrow$ All eigenvalues except the one corresponding to the $\mathbb{1}$ are < 1.

Recall a D-normalized matrix has D entries in every row and column that sum to one. To prove the above claim, we focus on the following properties of M_G:

If G is a connected D-regular graph:

- $\langle v, \mathbb{1} \rangle = 0$ for any eigenvector $v \neq \mathbb{1}$. For symmetric matrices, the eigenvectors are always orthogonal to one another.
- $M_G \cdot v = \lambda \cdot v$ for every eigenvector v for some eigenvalue λ

We use these facts to demonstrate that all other eigenvalues are less than 1 in M_G.

3.1.1 Graph Connectivicty Properties

If a graph G is connected, we can partition the vertices of the graph into to disjoint sets P and N such that there is an edge connecting at least one vertex from P to a vertex in N. Therefore, we split any eigenvector into two sets as follows:

- $P = \{i : v_i \geq 0\} \neq \emptyset$
- $N = \{i : v_i < 0\} \neq \emptyset$

In other words, P is the set of all vector entries that are non-negative and N is the set of all vector entries that are negative. Recall that:

$$\langle \mathbb{1}, v \rangle = 0$$

Which means that the sum of all entries of the eigenvector must be 0. Since v is not the zeros vector, we can infer that some entries of v must be posititve and some must be negative. This allows us to make the above claim that P and N are both nonempty sets.

3.1.2 Showing the Contradiction

We now show that any eigenvalue must be less than 1. First we rewrite the equation as follows:

$$\begin{aligned} (M_G \cdot v)_i &= \lambda \cdot v_i \\ \lambda \cdot v_i &= \sum_j M_G[i,j] \cdot v_j \end{aligned}$$

We consider the sum of all of the positive entries of the matrix, following the eigen value multiplication we get:

$$\lambda \sum_{i \in P} v_i = \sum_{i \in P} \sum_j M_G[i,j] v_j$$

In other words, if we take the sum of all positive entries in v, and multiply them by the vector's eigenvalue, we should get the same value when summing all the dot products of the eigenvector with the rows of M_G that correspond to positive entries in v. Observe that since we are taking a summation over terms, we are allowed to rewrite the summation relatively freely. We rewrite the above as follows:

$$\begin{aligned}\lambda \sum_{i \in P} v_i &= \sum_{j=1}^{n} v_j \sum_{i \in P} M_G[i,j] \\ &= \sum_{j \in P} v_j \sum_{i \in P} M_G[i,j] + \sum_{j \in N} v_j \sum_{i \in P} M_G[i,j]\end{aligned}$$

Since the sum of all entries of a row is 1, we can deduce that the above summation is:

$$\leq \sum_{j \in P} v_j \cdot 1 + \sum_{j \in N} v_j \sum_{i \in P} M_G[i,j]$$

Furthermore, since there is at least one edge connecting P to N, we know that at least one $M_G[i,j]$ is non-zero, meaning that the second part of the summation must account for some negative value. We can therefore conclude that the above summation must be strictly

$$< \sum_{j \in P} v_j \cdot 1$$

This demonstrates that the eigenvalue for any eigenvector v must be strictly less than one if $v \neq \mathbb{1}$ as the sum of the positive entries after multiplying the vector by the matrix must be strictly less than their original sum.

3.2 Theorem 2

We extend on the above proof with the following theorem:

Theorem 3.2 (SGT Theorem 2)**.** *G is a connected, regular, and has self-loops. Then:*

(a) 1 is an eigenvalue

(b) All other eigenvalues < 1 in absolute value. (Exercise)

The self loops just means that there are ones across the diagonal of the matrix.

4 Spectral Properties of Graphs

Theorem 4.1 (Eigenvalue Decomposition Theorem)**.** *Any symmetric matrix $M \in \mathbb{R}^{n \times n}$ has:*

- *n eigenvalues: $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$*
- *n eigenvectors: $v_1, v_2, ..., v_n$ that are orthonormal meaning $\langle v_i, v_i \rangle = 1$ and $\langle v_i, v_j \rangle = 0$ if $i \neq j$*
- *$M = \lambda_1 \cdot v_1 \cdot v_1^T + \lambda_2 \cdot v_2 \cdot v_2^T + ... + \lambda_n \cdot v_n \cdot v_n^T$*

If G is regular, then M_G has eigenvalues:

$$1 = \lambda_1(G) \geq \lambda_2(G) \geq ... \geq \lambda_n(G) \geq -1$$

Theorem 4.2 (SGT Theorem 2)**.** *G is regular and has self loops $\Rightarrow$* $\max(|\lambda_2(G)|, |\lambda_3(G)|, ..., |\lambda_n(G)|) < 1$

We define the following to use in our proof. If G is regular,

$$\lambda(G) = \max(|\lambda_2(G)|, |\lambda_3(G)|, ..., |\lambda_n(G)|)$$

and the "Spectral Gap of G" $= 1 - \lambda(G)$

Theorem 4.3. *SGT Theorem 2: G is regular, has self loops, and is connected $\iff$ Spectral Gap of $G > 0$.*

We care about the spectral gap of a graph G cause it represents how connected a graph is. The more connected a graph is, the spectral gap will be close to one with disconnected graphs having gaps close to 1.

Notation: G is a (N, D, λ) graph (has self-loops) if:

- G has N vertices
- G is $D-$regular
- The Spectral gap $\geq 1 - \lambda$ or $\lambda(G) \leq \lambda$.

Theorem 4.4 (SGT Theorem 3:)**.** *G is a (N, D, λ)-graph $\implies$ for any two vertices s, t in G, the shortest path between them $\leq \lceil \log_{1/\lambda} N \rceil + 1$.*

For example, if $\lambda \leq 1/2$, any two vertices are connected by $\leq \lceil \log_2 N \rceil + 1$ path. We do not prove this theorem but we have the following sketch:

4.0.1 Sketch of SGT Theorem 3

Consider the Eigenvalue Decomposition representation of M_G:

$$M_G = 1 \cdot \mathbb{1} \cdot \mathbb{1}^\top + \lambda_2 \cdot v_2 \cdot v_2^\top + ... + \lambda_N \cdot v_N \cdot v_N^\top$$

Now, when we take powers of a matrix, the eigenvalues get magnified:

$$M_G^k = 1 \cdot \mathbb{1} \cdot \mathbb{1}^\top + \lambda_2^k \cdot v_2 \cdot v_2^\top + ... + \lambda_N^k \cdot v_N \cdot v_N^\top$$

Now we have a sum of matrices where the first matrix is a bunch of $1/\sqrt{N}$ entries and the other terms decay exponentially since the λ values should always be low. If we have $\lambda = \frac{1}{2}$ and $k = 2 \log N$, the lambda becomes less than $1/N^2$. For any specific entry, we have:

$$\frac{1}{N} \pm (\frac{1}{N^2} \pm \frac{1}{N^2} + ...)$$

We can therefore say that $M_G^k > 0$ for all $M_G^k[i, j]$.

4.0.2 Diameter of Graph

We define $\Delta(G)$ as:

$$\delta(G) \ = \ \begin{cases} \infty, & \text{if } G \text{ is disconnected,} \\ \max_{s,t \in V(G)} \operatorname{dist}_G(s,t), & \text{otherwise,} \end{cases}$$

Expander Graphs: $\lambda \leq 1/2$ implies Diameter is small. Also implies lots of "redundancies" meaning any subset of vertices has lots of edges connecting them. More formally:

$$\forall S, |S| < 0.1n, \text{ and lots of edges leave } S$$

4.1 USTCON Revisited

Consider the case where we are promised that the connected components of G have small diameter.

$$\Delta(\text{Components}) \leq T$$

Now we know that either s, t are not connected or that there is a path of length $\leq T$. Using this, we use the brute force algorithm to solve $s - t$ connectivity as follows.

1. Explore all paths of lengtth $\leq T$ from s
2. If you reach t in these explorations we output YES.
3. If Not, output NO

The way we do this is simply having an algorithm that enumerates an ordering of the neighbors of each vertex.

Graph G is represented as:

- Vertex 1: [...]
- Vertex 2: [...]
- Vertex 3: [...]
- Vertex 4: [...]

The algorithm looks something like this:

Algorithm 2 Reachability via bounded walks

```
current ← S
for each sequence (i_1, i_2, ..., i_T) ∈ [D]^T do
    current ← S
    for ℓ = 1 to T do
        current ← the i_ℓ-th neighbor of current
        if current = t then
            return YES
        end if
    end for
end for
return NO
```

Total memory:

- $(s$, current, $t)$: $O(\log N)$
- Sequence: $T \cdot \log_2 D$
- l: $\log T$ bits

Memory: $O(\log N + T \log_2 D)$ bits

Time: $O(D^T \cdot T)$

So we can solve USTCON problem with the above space and time complexity. Note that if we can reduce a graph with logarithmic diameter and constant degree, we have generated an algorithm that solves USTCON in logspace and polynomial time. To do this, we are going to make it so every connected component is extremely well connected.

Target: Reduce to the case where:

- D is a constant
- Every connected component has $\lambda(G) \leq \frac{1}{2}$

Brute force alg: Memory: $O(\log N)$, Runtime: $N^{O(1)}$

Reingold's Idea/Algorithm
$(G, s, t) \rightsquigarrow (\bar{G}, \bar{s}, \bar{t})$ where s, t are connected in $G \Leftrightarrow \bar{s}, \bar{t}$ are connectd in $\bar{G}$

- $\lambda(\bar{G}) < \lambda(G)$
- Degree$(\bar{G}) \approx$ Degree(G)

The general reduction to reduce degree of G to $\bar{G}$ where $\bar{G}$ is a $4-$regular graph. We do so by increasing the number of vertices and decreasing the degree. For each edge, we break each old edge into two vertices at the endponits. Connect these new vertices so each vertex in G gets D new vertices to represent it, and each of those vertices are conected in a cycle. The degree of these new vertices is now 3 and we add a self-loop to make it 4.

5 Reducing General Graphs (Lecture 3)

Goal: Need operations that are computable in logspace to improve the diameter (spectral gap), reduce the second largest eigenvalue.

Diameter: $O(\log_{1/\lambda} N)$

We can decrease the eigenvalues $\lambda_2, ..., \lambda_n$ by increasing the connectivity of the graph. We add "shortcut" edges that add edges from any vertex to all

vertices distance 2 away. This will decrease the diameter by at least a factor of 2. We generate this by squaring the graph, taking adjaceny matrix A_G and generating $\bar{G} = A_G^2$. We have the following:

$$(N, D, \lambda) \rightsquigarrow (N, D^2, \lambda^2)$$

This follows as the degree increases by a multiplicative factor of D since there should be D neighbors of each of vertices D neighbors.

Squaring: If $(u, v)\&(v, w)$ are edges, add new edge (u, w)

Lemma 5.1. If G is a (N, D, λ) graph and G^2 is a (N, D^2, λ^2) graph. The brute force search yields the following memory requirement:

- G: $\log_2 N + \log D \cdot \log_{1/\lambda} N$
- G^2: $\log_2 N + \log D^2 \cdot \log 1/\lambda^2 N$

We aim to find a "powering operation" that improves the spectral gap while not blowing up the degree as much.

5.1 Reingold's Algorithm

Select H: A special graph generated from

$$(G, s, t) \rightsquigarrow (G^2 \circledcirc H, \bar{s}, \bar{t})$$

The main idea is squaring G to decrease λ, and then applying the $\circledcirc$ operation to decrease the degree. We repeat the above operation until we get a graph we can use to perform the algorithm. We define this recursively as follows:

$$G_{i+1} = G_i^2 \circledcirc H$$

6 Zig-Zag Product

We need the following to hold:

- G is a (N, D, λ_G)-graph
- H is a (D, D_1, λ_H)-graph

We use something called **Consistent Labeling**. For a D regular graph, a consistent labeling is a mapping $L : [E] \to \{1, ..., D\}$ such that each vertex has all of its outgoing edges getting distinct labels.

Consider G as a $4-$regular graph consistently labeled. We generate H as a 2-regular graph on 4 vertices. The zig-zag product is applied as follows:

- Every vertex in G gets replaced with a copy of H.
- Degree of $G \circledcirc H$ is always going to be D_1^2.

To define this new degree D_1^2, we consider $[D_1] \times [D_1]$. Every edge in $G \circledcirc H$ corresponds to a pair of $(k_1, k_2) \in [D_1] \times [D_1]$. The edges are defined as follows:

- We take the edge labeled $k_1 = (u, v)$ from vertex u in H.
- From vertex v, we take the edge labled v in G to go to the corresponding cloud.
- We take a final step from the new cloud's vertex v to the vertex it is connected to by edge k_2. (I would look at notes for this) (Review how this works!)
- We only keep the edge from the starting u vertex from a copy of H to the final vertex

The algorithm for computing the edges is as follows. $H = (D, D_1, -)$ graph:

1. For every new vertex $(u, a)\&(k_1, k_2) \in [D_1] \times [D_1]$
2. Take a within cloud step according to k_1 to get to (u, b)
3. Take an across cloud step. Let v be the b^{th} neighbor of u in G. Jump to (v, b)
4. Take edge labeled k_2 from b within the cloud: (v, b').
5. This results in us connecting (u, a) to (v, b') with edge label (k_1, k_2).

Definition 6.1 (Rotation Map)**.** A **Rotation Map** of a graph $G : (N, D, -)$ is defined as:

$$Rot_G : [N] \times [D] \to [N] \times [D]$$

So $Rot(v, i) = (w, j)$ where w is the ith neighbor of v and v is the j^{th} edge out of w.

Given $G : (N, D, -)$-graph and $H : (D, D_1, -)-$graph, we get $G \circledcirc H : (ND, D_1^2, -)$

ZigZag takes as input:

- $Rot_G : [N] \times [D] \to [N] \times [D]$
- $Rot_H : [D] \times [D_1] \to [D] \times [D_1]$

And outputs $Rot_{G \circledcirc H} : [ND] \times [D_1^2] \to [ND] \times [D_1^2]$. We create this transformation by doing the following:

$Rot_{G \circledcirc H}((u, a), (k_1, k_2))$:

- $(b, k_1') \longleftarrow Rot_H(a, k_1)$ 1st cloud step
- $(v, c) \longleftarrow Rot_G(u, b)$ between cloud step
- $(d, k_2') \longleftarrow Rot_H(c, k_2)$ 2nd cloud step

Output: $((v, d), (k_2', k_1'))$

To understand what the above transformation is, we need to analyze what the original rotation as well as the sub rotations are asking for. To get a rotation map of a node in the new graph $G \circledcirc H$, we can conceptualize it as labelling the cloud corresponding to u, as well as the starting node corresponding to the edge of u that it is coming out of. For example, (u, a), corresponds to the cloud that represents node u in G and the corresponding node a in H that represents the a^{th} edge coming out of u. The second tuple $[D_1^2]$, corresponds to the edges in H that should be followed out of the cloud for u, say (k_1, k_2). As previously explained, the k_1^{th} edge out of the node that corresponds to a in H should be taken to the new edge in H, say node b. Now, we take the edge from that node b to the corresponding cloud of $v \in G$. From the edge labeled b in this cloud, we take the k_2^{th} edge to a new node say c. This is essentially what the above transformation is explaining. More succinctly, we analyze the final output $((v, d), (k_2', k_1')) \longleftarrow Rot_{G \circledcirc H}((u, a), (k_1, k_2))$. Firstly, we can see that we are going from the cloud reprsenting u to the cloud representing v so it makes sense that we end up in cloud v. To get the node d in H, we follow the path from a in the cloud for u. First, it takes the k_1^{th} edge in H to get to node b. It now has to take a step from this node b to the correct node in the cloud for v. We say that v is the b^{th} neighbor of u and u is the c^{th} neighbor of v. Because of this, the rotation mapping has to return the node representing the c^{th} neighbor

in cloud v. From here, we need to take the k_2^{th} edge from this c node to the final node d, which gives us the resulting (v, d) tuple. For the second tuple, we need to konw how to get from (v, d) back to (u, a). For this, we need the k_1', k_2' values that simulate the previously mentioned path back to (u, a). Following the logic, we need to go back from the d^{th} node in the cloud to the c^{th} node which would be the $k_2'^{th}$ neighbor of d, and from there we step back to the cloud for u where we are now at the node labeled b, which we then take its $k_1'^{th}$ edge to get back to a explaining the (k_2', k_1') part of the tuple.

Theorem 6.1 (Rozenman, Vodham 05, RVW01)**.** *$G \circledcirc H$ is a (ND, D^2, λ)-graph where.*

$$\lambda \leq 1 - (1 - \lambda_H)^2(1 - \lambda_G)$$

Intuitively, if G, H have small λ's, then $G \circledcirc H$ also has small λ and degree of $G \circledcirc H$ is smaller than degree of G. The degree is smaller and the spectral gap is slightly worse.

Lemma 6.2. If s, t are in disconnected parts of G, the clouds of s, t in $G \circledcirc H$ will still be disconnected.

7 Expander Graphs

There are numerous definitions for expander graphs that are all equivalent:

Definition 7.1. Consider a graph that has N vertices and is D-regular. The combinatorial definition of expansion is to say that for any big set of vertices S such that $S \subset [N], |S| \leq N/3$, the number of edges from S to $\bar{S}$ is at least $(0.4)D \cdot |S|$. So to disconnect S from the rest of the graph, we would need to cut at least that many edges.

Another definition for exander graphs that is equivalent is saying that the second largest eigenvalue in absolute value is less than or equal to 0.1. (These constants are made up). This also implies the diameter is very very small.

7.1 History

Until around 2001, we only had such graphs where D is a constant from deep number theoretic results. The construction of a more combinatorial

version of expander graphs is by starting with two small expanders G_0, H and generating new graphs as follows:

$$G_{i+1} = G_i^2 \circledcirc H$$

You can also tensor G_i.

8 Lecture 4, Main Spectral Theorem for ZigZag

Theorem 8.1 (Rozenmann, Vadhan 05, RVW01)**.** *$G \circledcirc H$ is a (ND, D_1^2, λ)-graph where:*

$$\lambda \leq 1 - (1 - \lambda_H)^2(1 - \lambda_G)$$

Or equivalently:

$$(1 - \lambda_H)^2(1 - \lambda_G) \leq 1 - \lambda$$

Let $F = G \circledcirc H$. F has $N \cdot D$ many vertices. To help visualize this, think of these ND vertices as blocks of D vertices. Each row/col in the final matrix can be though of as $(1, a), (1, b)..., (2, a), (2, b), ...(N, 1)..(N, D)$. The total matrix will be $[(N, D)] \times [(N, D)]$.

We take an "edge hop" from an edge according to H to and edge according to G to a final edge according to H. We can think of this as three matrices that are $N \times D$ in size:

- The left and rightmost matrices represents the within cloud steps. Each diagonal entry represents a copy of matrix H. This tells where to step within the cloud.
- The intercloud step matrix in the middle, which we will call P, represents the transition from a particular cloud to another. It is a permutation matrix that transitions a particular node (u, b) to the cloud representing the b^{th} neighbor of u say v.

We now have a very nice representation of matrix F as the product of three matrices. The Zig-Step can further be simplified as the tensor product of $(\mathbf{I}_n \otimes A_H) \cdot P_G \cdot (\mathbf{I}_n \otimes A_H)$. We write the normalized adjaceny matrix as:

$$\begin{aligned} A_F = \frac{A_F}{D_1^2} &= \mathbf{I}_N \otimes \frac{A_H}{D_1} \cdot P_G \cdot \mathbf{I}_N \otimes \frac{A_H}{D_1} \\ &= \mathbf{I}_N \otimes M_H \cdot P_G \cdot \mathbf{I}_N \otimes M_H \end{aligned}$$

Goal: $M_F = (\mathbf{I}_N \otimes M_H) \cdot P_G \cdot (\mathbf{I}_N \otimes M_H) \implies \lambda_F \leq 1 - (1-\lambda_H)^2 \cdot (1-\lambda_G)$ where λ_F is the second largest eighenvalue for M_F.

Lemma 8.2. G is an (N, D, λ) graph. We know that

- 1 is an eigenvalue and the all 1's vector is an eigenvector
- All other eigenvalues are less than λ in absolute value (definition)

$M_G = 1 \cdot \frac{1}{N}\mathbb{1}\mathbb{1}^\top + \lambda_2 \cdot v_2 \cdot v_2^t op + ... + \lambda_n \cdot v_n \cdot v_n^\top$ The first matrix is $\frac{1}{N}$ times the all 1's matrix. We rewrite

$$M_G = \frac{1-\lambda}{N}\mathbf{1}^{n\times n} + \frac{\lambda}{N}\mathbf{1}^{n\times n} + \lambda_2 \cdot v_2 \cdot v_2^\top$$

We view the terms that are not $\frac{1-\lambda}{N}\mathbf{1}^{n\times n}$ as error. Continuing from the lemma, we rewrite M_G as:

$$M_G = (1-\lambda)\frac{J_N}{N} + \lambda \cdot E$$

Where $||E|| \leq 1$. The **spectral norm** for any symmetric matrix, A, $||A|| =$ largest abs-value of an eigenvalue of A. We restate the lemma:
Lemma: A graph G is an (N, D, λ)-graph $\iff$ it can be rewritten as mentioned above.

To prove the first direction, suppose we rewrite the graph as:

$$M_G = \frac{1-\lambda}{N}\mathbf{1}^{n\times n} + \frac{\lambda}{N}\mathbf{1}^{n\times n} + \lambda_2 \cdot v_2 \cdot v_2^\top ...$$

We know that the second through nth terms form another matrix E, so we rewrite it as:

$$\frac{1-\lambda}{N} \cdot J + E$$

We know that $||E|| \leq \lambda$ since it is an (N, D, λ) graph by definition. The other direction is not gonna be proven!