# Summary Questions of the lecture

- Explain the key aspects of GAT: Graph Attention Networks.

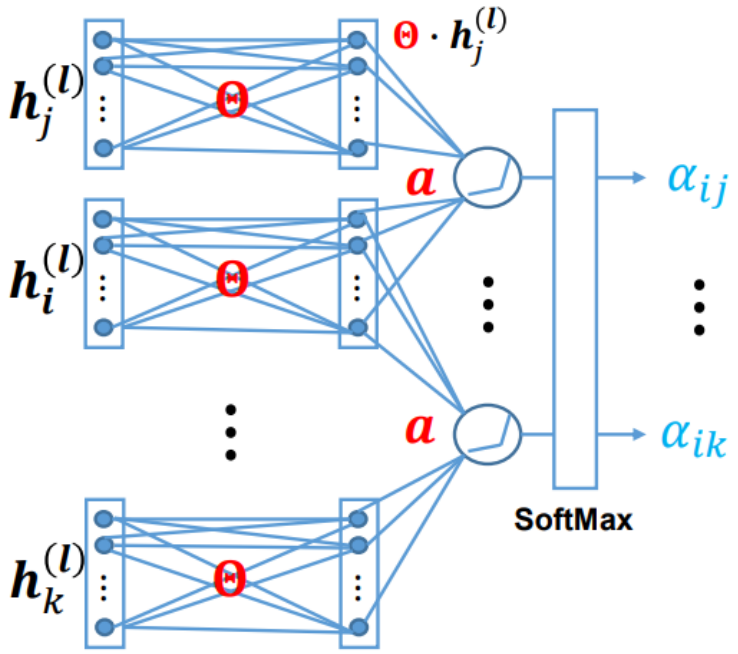
→ GAT aggregates the transformed feature vectors of neighbor nodes $\Theta h_j^{(l)}$ by weighting them with attention. The attention weights are generated by applying softmax across the compatibility scores of neighbor nodes: $LeakyReLU(a^T[\Theta h_i^{(l)} || \Theta h_j^{(l)}])$. Note that $a$ and $\Theta$ are learnable parameters.

**Aggregation:**

$$\boldsymbol{h_i^{(l+1)} = \sigma\left(\sum_{v_j \in N(v_i)} \alpha_{ij}\, \Theta \cdot h_j^{(l)}\right)}$$

$$\alpha_{ij} = \frac{exp\left(LeakyReLU\left(a^T\left[\Theta \cdot h_i^{(l)} \,\|\, \Theta \cdot h_j^{(l)}\right]\right)\right)}{\sum_{v_k \in N(v_i)} exp\left(LeakyReLU\left(a^T\left[\Theta \cdot h_i^{(l)} \,\|\, \Theta \cdot h_k^{(l)}\right]\right)\right)}$$

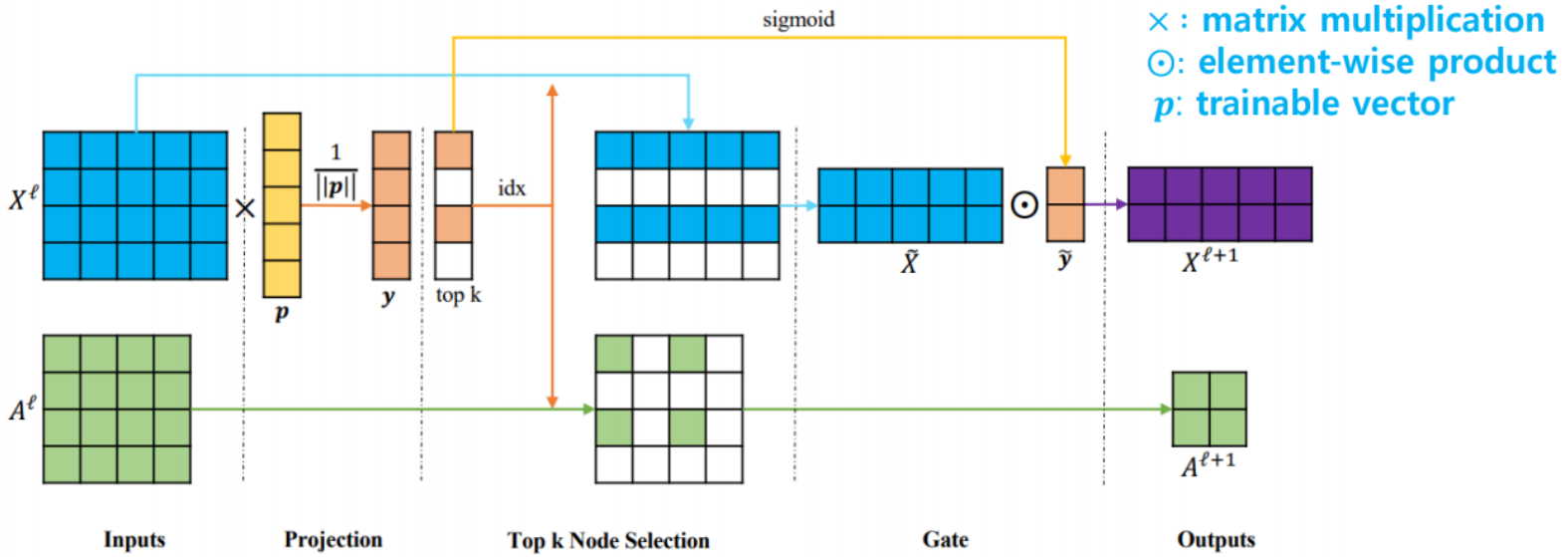
$a, \Theta$: parameters of a single layer network



# Summary Questions of the lecture

- Explain the key aspects of gPool: Graph U-Nets.

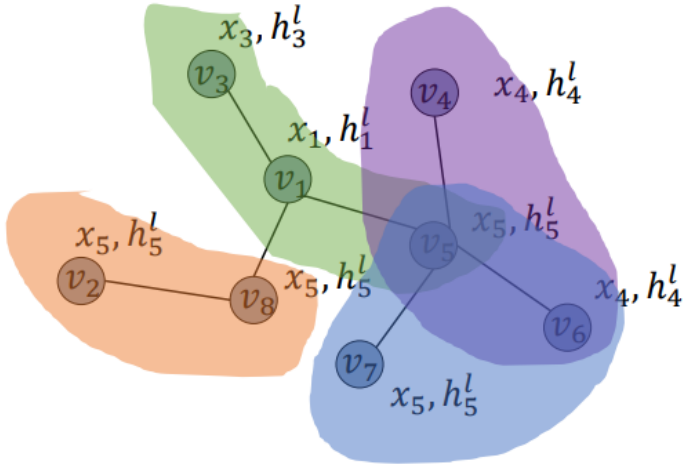
→ Graph U-Nets downsamples the graph using gPool, upsamples it using gUnpool, and transforms graph features with the simplified ChebNet. Specifically, gPool downsamples a given graph $(X^l, A^l)$ by selecting the top $k$ most important nodes. The importance scores $y$ are generated by multiplying a learnable parameter vector ($1 \times 1$ convolution filter) $p$ to $X^l$ and normalizing it. The scores are also used to gate the graph signals of the selected nodes to generate the output graph signals. gUnpool simply restores the graph back to its previous structure.



# Summary Questions of the lecture

- Explain the key aspects of DiffPool: Hierarchical Graph Representation Learning with Differentiable Pooling

→ DiffPool first embeds the input graph $X$ to $H$ through the simplified ChebNet operation. Then, it generates a soft assignment matrix $S$ through another simplified ChebNet operation, but this time its output is normalized with the softmax. Since $S$ provides the probability matrix for each node being assigned to each cluster, the input graph can be pooled with $H_p = S^T H$ and $A_p = S^T AS$.




Assignment Matrix for pooling: $\boldsymbol{S} \in \mathbb{R}^{n \times n_p}$

$$\boldsymbol{S} = \boldsymbol{SoftMax}\left(\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}\boldsymbol{X}\boldsymbol{\Theta}_{\boldsymbol{s}}\right) \leftarrow \boldsymbol{GCN}$$

GCN Filtering(node embedding): $\boldsymbol{H} \in \mathbb{R}^{n \times d_p}$

$$\boldsymbol{H} = \boldsymbol{ReLU}\left(\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}\boldsymbol{X}\boldsymbol{\Theta}_{\boldsymbol{h}}\right) \leftarrow \boldsymbol{GCN}$$

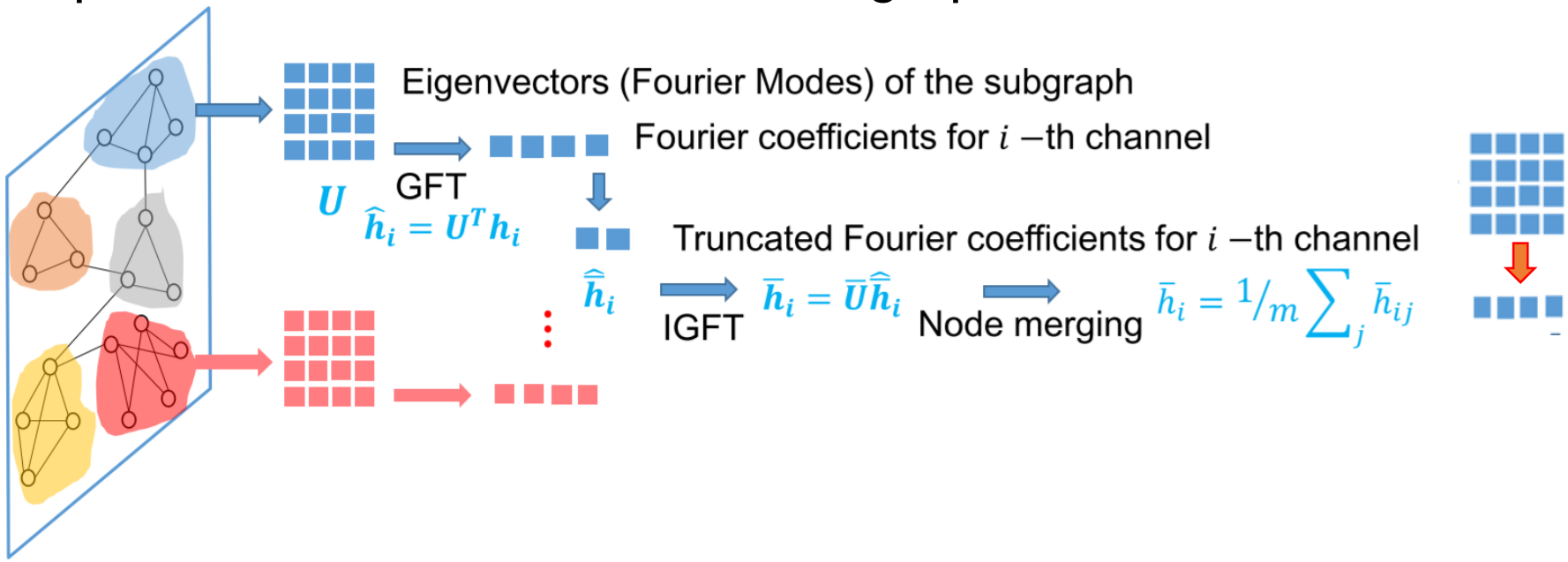
DiffPool layer:

$$\boldsymbol{H_p} = \boldsymbol{S^T H} \in \mathbb{R}^{n_p \times d_p}$$

$$\boldsymbol{A_p} = \boldsymbol{S^T AS} \in \{0,1\}^{n_p \times n_p}$$

# Summary Questions of the lecture

- Explain the key aspects of EigenPooling: Graph Convolutional Networks with EigenPooling
- EigenPooling first clusters the input graph by directly adopting Laplacian clustering. Then, within each cluster's sub-graph, the node features are smoothed by applying a hard low pass filter in the spectral domain. The resulting sub-graph feature vectors are averaged to obtain the feature vector of a representative node for each sub-graph.



# Outline of Lecture (4)

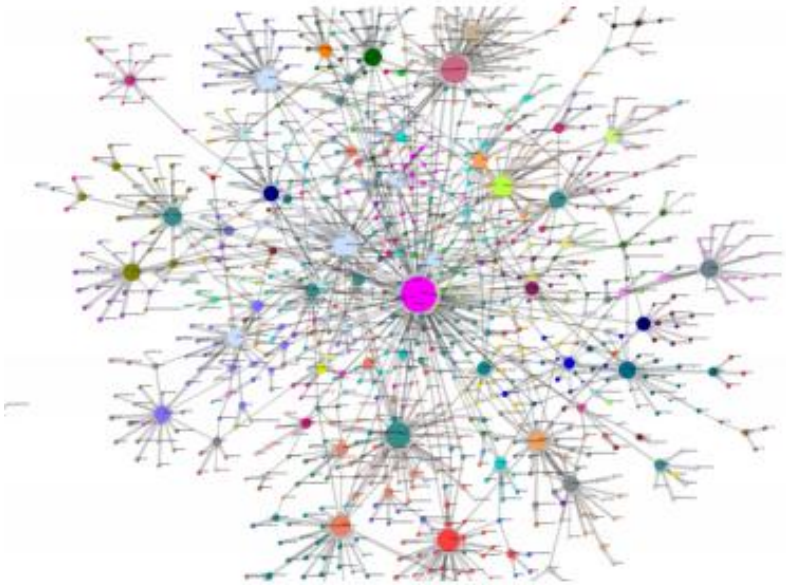
- Spatial GCN
  - Spatial View of Simplified ChebNet
  - GraphSage (Hamilton et al. NIPS 2017)
  - GAT : Graph Attention (Veličković et al. ICLR 2018)
  - MPNN: Message Passing (Glimer et al. ICML 2017)
  - **gPool:** Graph U-Nets **(**Gao et al. ICML 2019)
  - **DiffPool:** Differentiable Pooling **(**Ying et al. NeurIPS 2018)
  - **EigenPooling:** EigenPooling (Ma et al. KDD 2019)

- Link Analysis
  - Directed Graph
    - Strongly Connected Graph
    - Directed Acyclic Graph
  - Link Analysis Algorithms
    - PageRank (Ranking of Nodes)
    - Personalized PageRank
    - Random Walk with Restarts

- Propagation using graph diffusion
  - Predict Then Propagate [ICLR'19]
  - Graph Diffusion-Embedding Networks [CVPR'19]
- Making a new graph
  - Diffusion Improves Graph Learning [NIPS'19]
  - Graph Learning-Convolutional Nets. [CVPR'19]

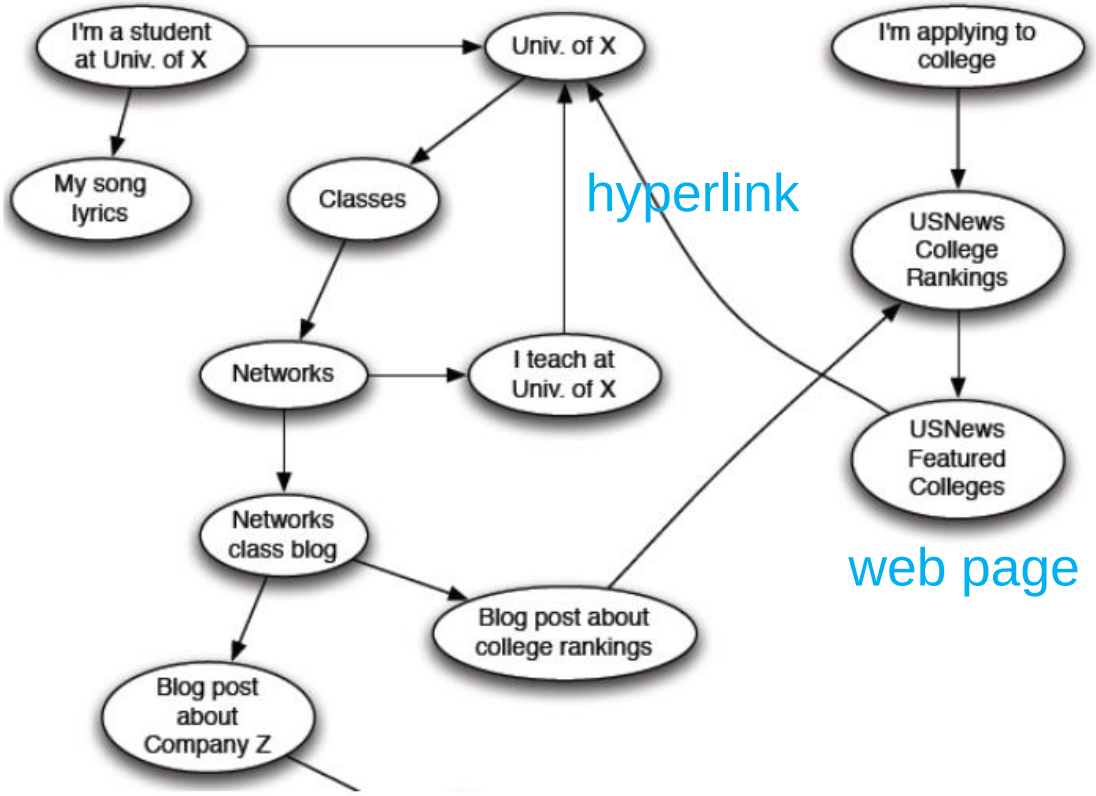
# *Link Analysis*

*PageRank, Directed Graph (Web, Epidemic),*
*Random work, Teleporting, ….*

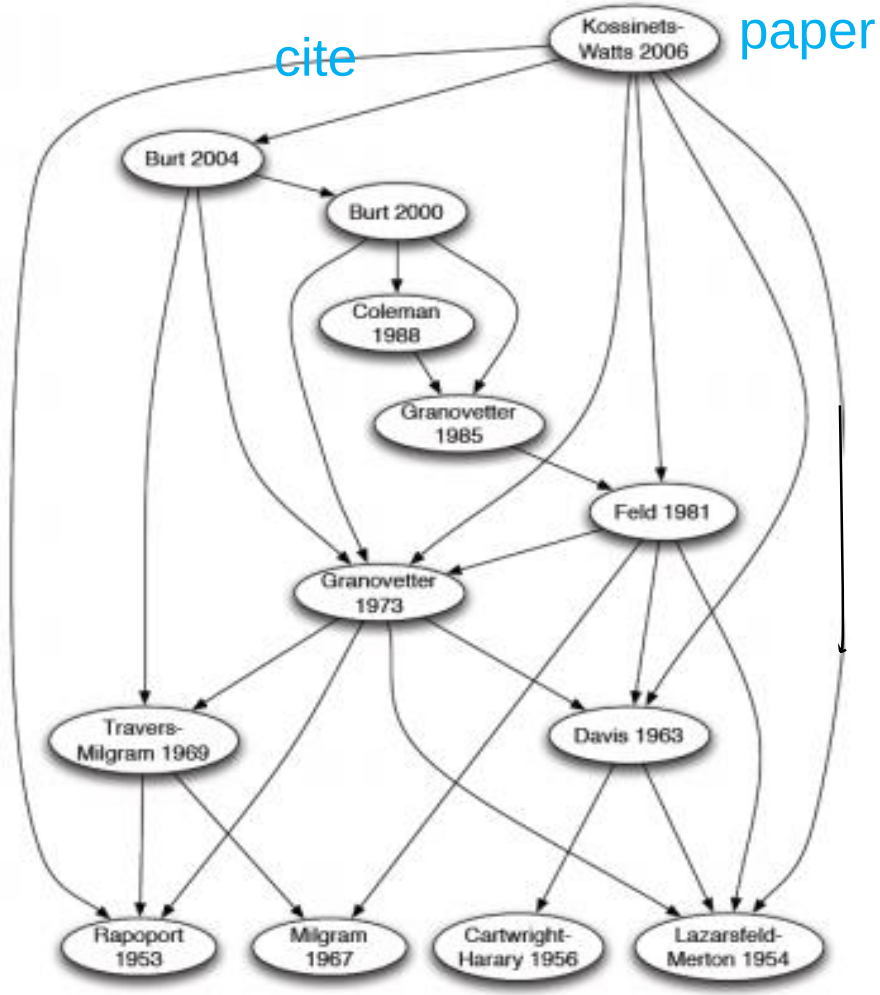


- Machine Learning with Graphs, Jurij Leskovec, Stanford University
- Recent papers

# Directed Graph




Web Pages

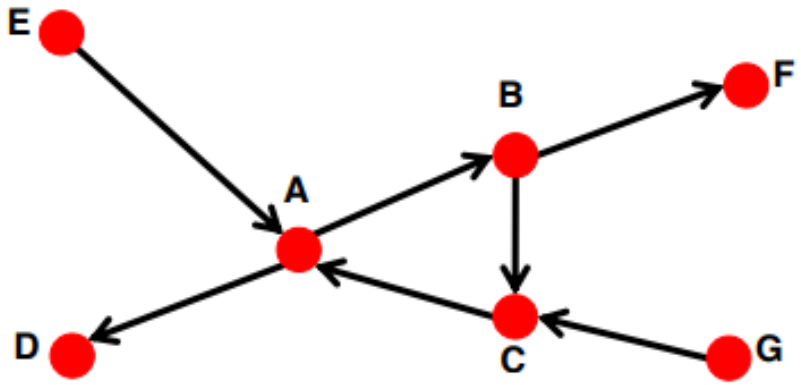



Citations

# Directed Graph

Directional Node Set: $In(v), Out(v)$

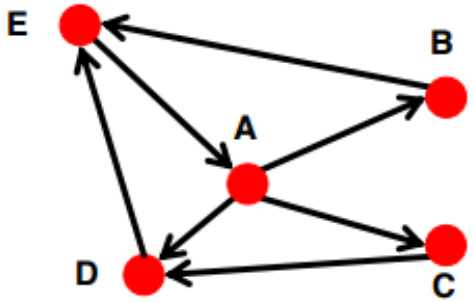
$$In(v) = \{w | w\ can\ reach\ v\}$$
$$Out(v) = \{w | v\ can\ reach\ w\}$$




$$In(A) = \{w | A, B, C, E, G\}$$
$$Out(A) = \{w | A, B, C, D, F\}$$

Graph structure in the Web, computer networks (Broder et al. 2000), 4078 cites

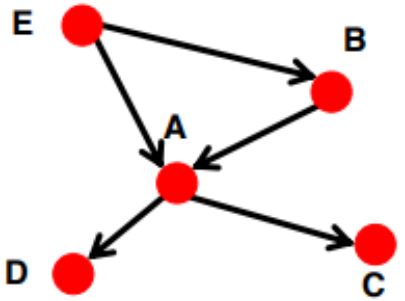
# Directed Graph




Two types of directed graphs:

- Strongly connected:
  - Any node can reach any node via a directed path

    In(A)=Out(A)={A,B,C,D,E}
- Directed Acyclic Graph (DAG):
  - Has no cycles: if $u$ can reach $v$, then $v$ cannot reach $u$.



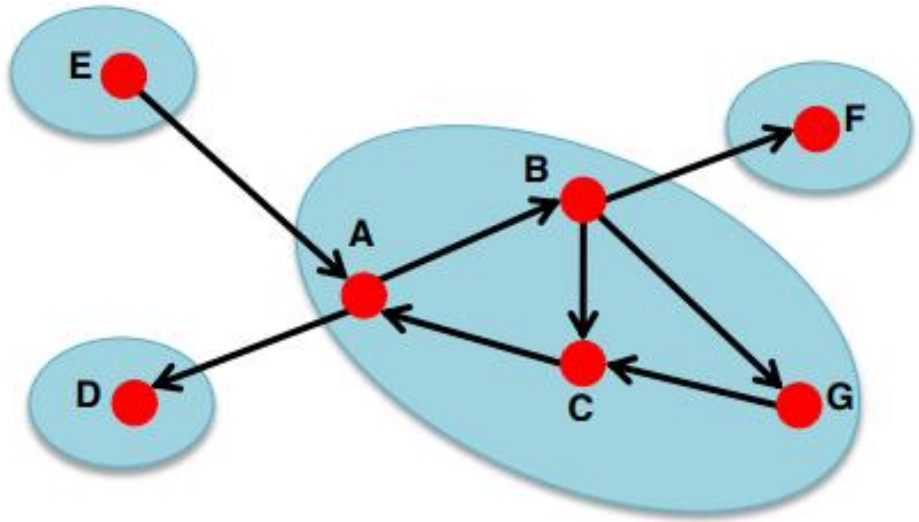

Any directed graph (the Web) can be expressed in terms of these two types!

- Is the Web a big strongly connected graph or a DAG? No.

# Directed Graph

A Strongly Connected Component (SCC) in a graph is a set of nodes $S$ so that:

- Every pair of nodes in $S$ can reach each other
- There is no larger set containing $S$ with this property




Strongly Connected Components in the graph:

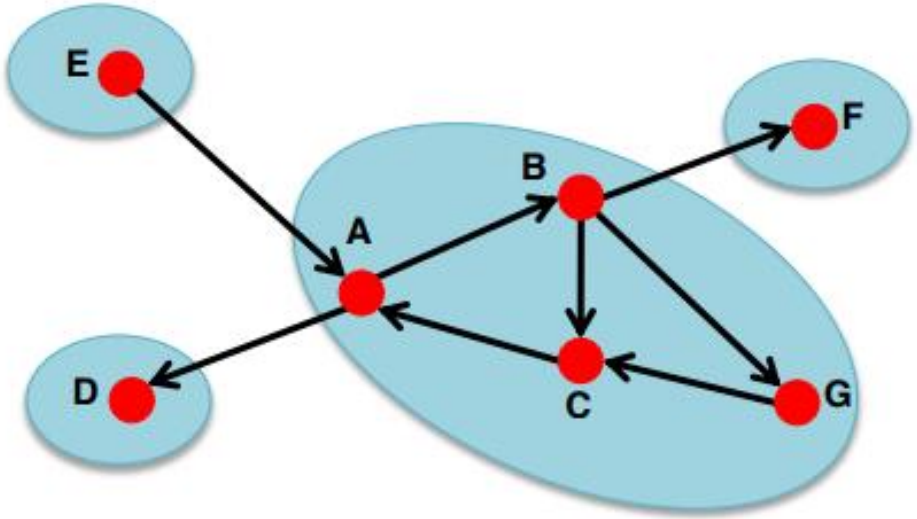
$$\{A, B, C, G\}, \{D\}, \{E\}, \{F\}$$

# Directed Graph

Fact: Every directed graph is a DAG on its SCCs

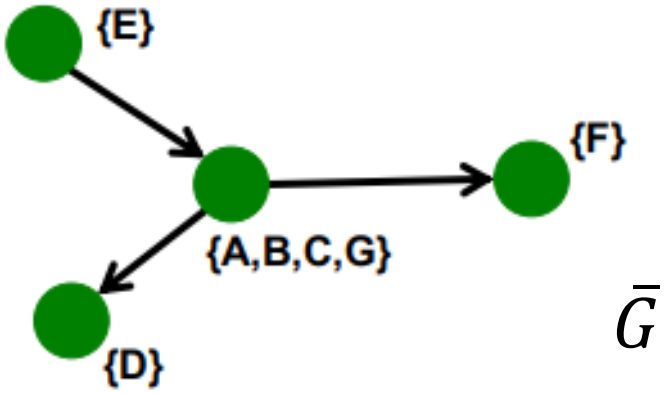
① SCCs partitions the nodes of $G$, where each node is in exactly one SCC

② If we build a graph $\bar{G}$ whose nodes are SCCs, and with an edge between nodes of $\bar{G}$ , and there is an edge between any two of SCCs in $G$, then $\bar{G}$ is a DAG




① Strongly Connected Components in the graph $G$: $\{A, B, C, G\}, \{D\}, \{E\}, \{F\}$

② $\bar{G}$ is a DAG




$\bar{G}$

# Directed Graph

Computational issue:

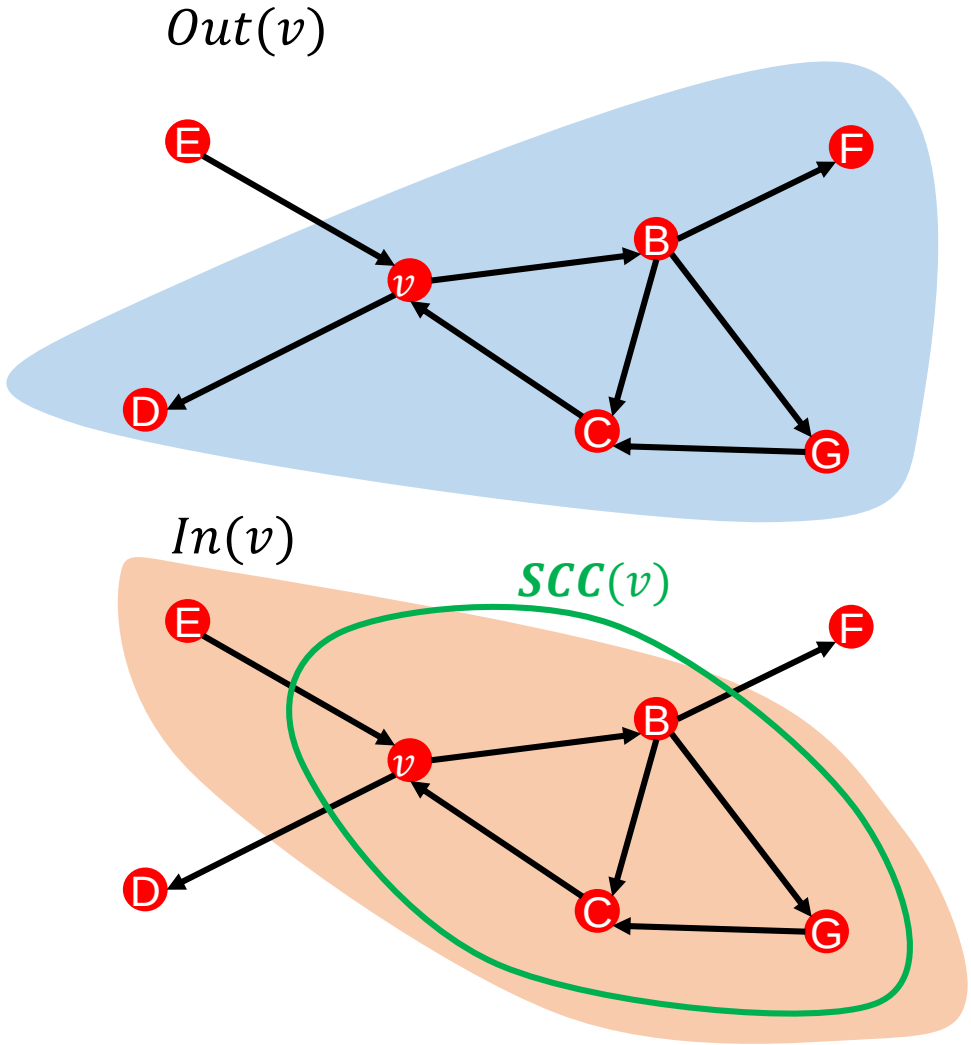
- Want to find a SCC containing node $v$?

Observation:

- $Out(v)$ … nodes that can be reached from $v$
- $In(v)$ … nodes that can reach to $v$
- SCC containing $v$ is:

$$Out(v) \cap In(v) = Out(v, G) \cap Out(v, \check{G}),$$

where $\check{G}$ is $G$ with all edge directions flipped



# Directed Graph

Altavista web crawl, Graph structure in the Web, Broder et al., 2000

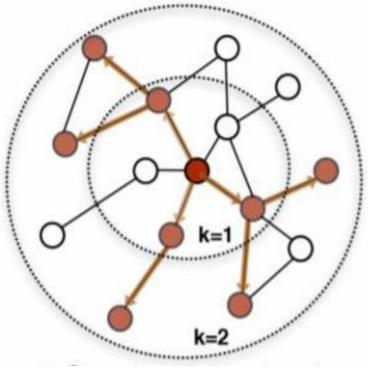
- 203 million URLs, 1.5 billion links

Computation:

- Compute $In(v)$ and $Out(v)$ by starting at random nodes.
- Observation: The BFS(breadth-first search) either visits many nodes or very few.

Result: Based on $In$ and $Out$ of a random node $v$:

- $Out(v)$ ≈ 100 million (50% nodes)
- $In(v)$ ≈ 100 million (50% nodes)
- Largest $SCC$: 56 million (28% nodes)




What does this tell us about the conceptual picture of the Web graph?

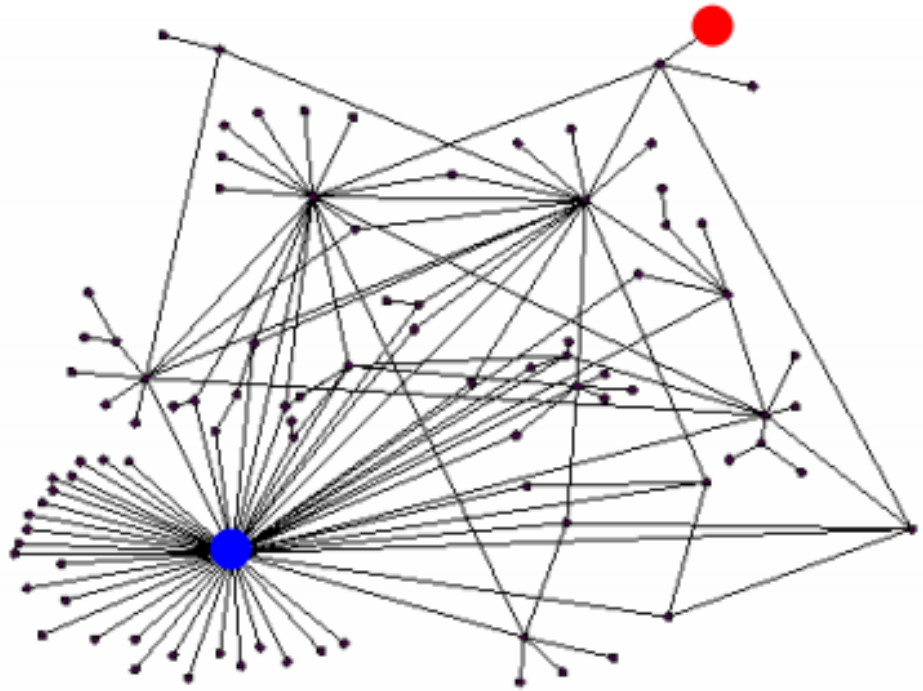
# Ranking Nodes on the Graph

All web pages are not equally “important”

Harmful Site vs. Seoul National University

There is large diversity in the web-graph node connectivity.

So, let’s rank the pages using the web graph link structure!



# Link Analysis Algorithms

We will cover the following Link Analysis approaches to compute the importance of nodes in a graph:

- PageRank
- Personalized PageRank
- Random Walk with Restarts

# Link as Votes

Idea: Links as votes

- Page is more important if it has more links
- In-coming links? Out-going links?

Think of in-links as votes:

- Seoul National University has 20,000 in-links per day
- Harmful Site has 2 in-links per day

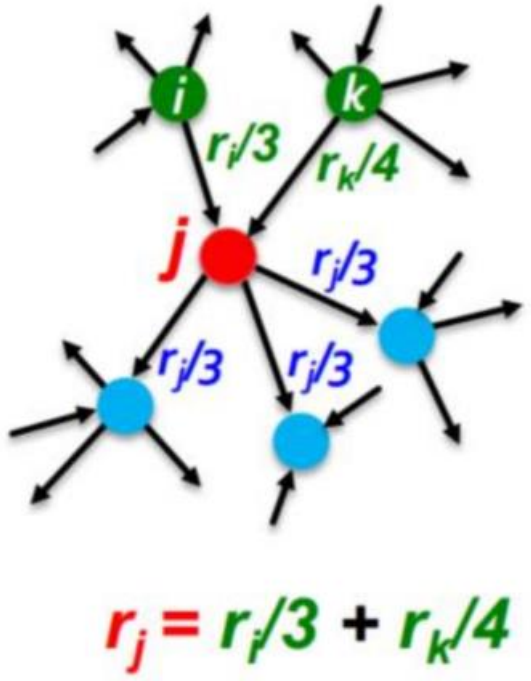
Are all in-links equal?

- Links from important pages count more
- Recursive question! (Importance propagation)

# PageRank: The ‘Flow’ Model

A “vote” from an important page is worth more:

- Each link’s vote is proportional to the importance of Its source page
- If page $i$ with importance $r_i$ has $d_i$ out-links, each link gets $r_i/d_i$ votes
- Page $j$’s own importance $r_j$ is the sum of the votes on its in links



# PageRank: The 'Flow' Model

- A page is important if it is pointed by other important pages
- Define a "rank" $r_j$ for node $j$

$$r_j = \sum_{i \to j} \frac{r_i}{d_i},$$

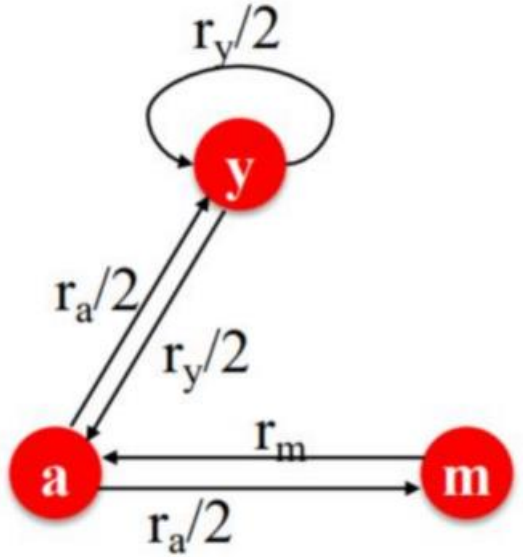
where $d_i$ is out-degree of node $i$.

- 'Flow' equations:

$$r_y = \frac{r_y}{2} + \frac{r_a}{2} \quad \rightarrow \quad r_y = r_a$$

$$r_a = \frac{r_y}{2} + r_m \quad \rightarrow \quad r_a = 2r_m$$

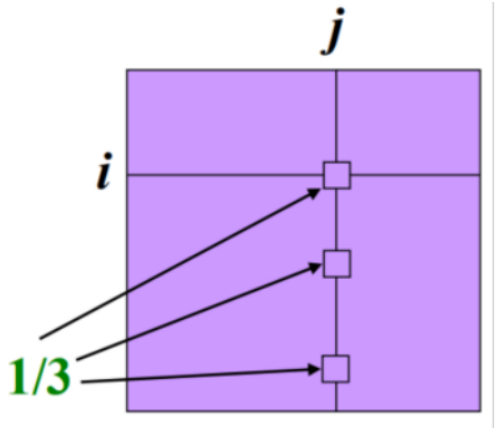
$$r_m = \frac{r_a}{2} \quad \rightarrow \quad r_m = \frac{r_a}{2}$$




Gaussian elimination to solve this system of linear equations.
Bad idea!

# PageRank: Matrix Formulation

- Stochastic adjacency matrix $\boldsymbol{M}$
  - Let page $j$ have $d_j$ out-links
  - If $j \rightarrow i$ , then $\boldsymbol{M}_{ij} = 1/d_j$
  - $\boldsymbol{M}$ is a column stochastic matrix of which Columns sum to 1
- Rank vector : An entry per page
  - $r_i$ is the importance score of page
  - $\sum_i r_i = 1$
- The flow equations can be written
  - $\boldsymbol{r} = \boldsymbol{M}\boldsymbol{r}$
  - $r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$




- 'Flow' equations:

$$r_y = \frac{r_y}{2} + \frac{r_a}{2}$$

$$r_a = \frac{r_y}{2} + r_m \rightarrow \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$$

$$r_m = \frac{r_a}{2}$$

# PageRank: Eigenvector Formulation

- The flow equations can be written
  - $\boldsymbol{r} = \boldsymbol{Mr}$

$$\boldsymbol{Ax} = \boldsymbol{\lambda x}$$

- So the rank vector $\boldsymbol{r}$ is an eigenvector of the stochastic web matrix $\boldsymbol{M}$ for eigenvalue $\lambda = 1$.
  - Starting from any vector $\boldsymbol{u}$, the limit $\boldsymbol{M}(\boldsymbol{M}(\ldots \boldsymbol{M}(\boldsymbol{Mu})))$ is the long-term distribution of the surfers.
  - The math: Limiting distribution = eigenvector for $\lambda = 1$ of **M** = PageRank.
  - Note: If $\boldsymbol{r}$ is the limit of $\boldsymbol{MM}\ldots \boldsymbol{MMu}$, then $\boldsymbol{r}$ satisfies the equation $\boldsymbol{r} = \boldsymbol{Mr}$, so $\boldsymbol{r}$ is an eigenvector of $\boldsymbol{M}$ with eigenvalue 1
- We can now efficiently solve for $\boldsymbol{r}$
- The method is called 'Power iteration'

# PageRank: Eigenvector Formulation

- Given a web graph with $N$ nodes, where the nodes are pages and edges are hyperlinks

- Power iteration: a simple iterative scheme
  - Initialize: $\boldsymbol{r}^{(0)} = \left[\frac{\mathbf{1}}{\boldsymbol{N}}, \ldots, \frac{\mathbf{1}}{\boldsymbol{N}}\right]^{\boldsymbol{T}}$.
  - Iterate: $\boldsymbol{r}^{(t+1)} = \boldsymbol{M}\boldsymbol{r}^{(t)}$
  - Stop when $\left\|\boldsymbol{r}^{(t+1)} - \boldsymbol{r}^{(t)}\right\|_1 < \epsilon$

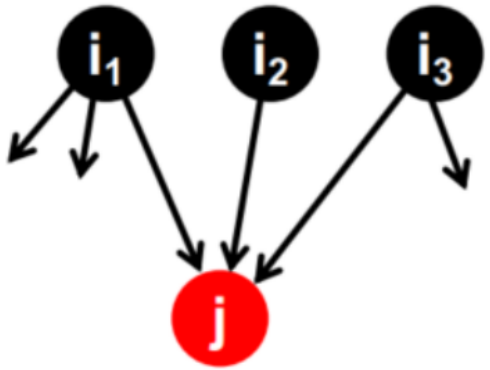
    where $\|\boldsymbol{x}\|_1 = \sum_{1 \le i \le N} |x_i|$ and we can use any other vector norms.

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

- About 50 iterations are sufficient to estimate the limiting solution

# PageRank: Random Work Interpretation

- Imagine a random web surfer:
  - At any time $t$, surfer is on some page $i$
  - At time $t+1$, the surfer follows an out-link from $i$ uniformly at random
  - Ends up on some page $j$ linked from $i$
  - Process repeats indefinitely
- Let:
  - $\boldsymbol{r}^{(t)}$ is a vector whose $i$ −th coordinate is the probability that the surfer is at page $i$ at time $t$
  - So, $\boldsymbol{r}^{(t)}$ is a probability distribution over pages




$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

# PageRank: How to solve?

R: Given a web graph with $N$ nodes, where the nodes are pages and edges are hyperlinks

- Initialize: $\boldsymbol{r}^{(0)} = \left[\frac{\mathbf{1}}{\boldsymbol{N}}, \ldots, \frac{\mathbf{1}}{\boldsymbol{N}}\right]^{\boldsymbol{T}}$.
- Iterate: $\boldsymbol{r}^{(t+1)} = \boldsymbol{M}\boldsymbol{r}^{(t)}$ $\leftarrow$ $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
- Stop when $\left\|\boldsymbol{r}^{(t+1)} - \boldsymbol{r}^{(t)}\right\|_1 < \epsilon$

  where $\|\boldsymbol{x}\|_1 = \sum_{1 \le i \le N} |x_i|$ and we can use any other vector norms.

- Does this converge?
- Does it converge to what we want?
- Are results reasonable?

# PageRank: Problems

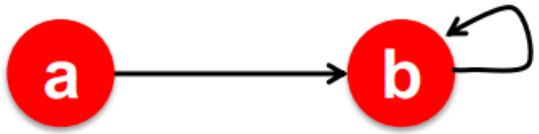
Two problems:

(1) Some pages are dead ends
- Such pages cause importance to “leak out”

(2) Spider traps (all out-links are within the group)
- Eventually spider traps absorb all importance

# PageRank: Does this converge?

The “Spider trap” problem:




$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

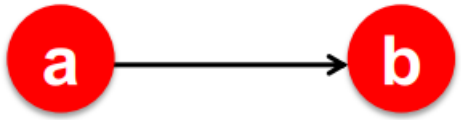
Example:

- Iteration → 0 | 1 | 2 | 3 ……

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $r_a$ | 1 | 0 | 0 | 0 |
| $r_b$ | 0 | 1 | 1 | 1 |

# PageRank: Does it converge to what we want?

The “Dead end” problem:




$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

Example:

- Iteration → 0 | 1 | 2 | 3 ……

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $r_a$ | 1 | 0 | 0 | 0 |
| $r_b$ | 0 | 1 | 0 | 0 |

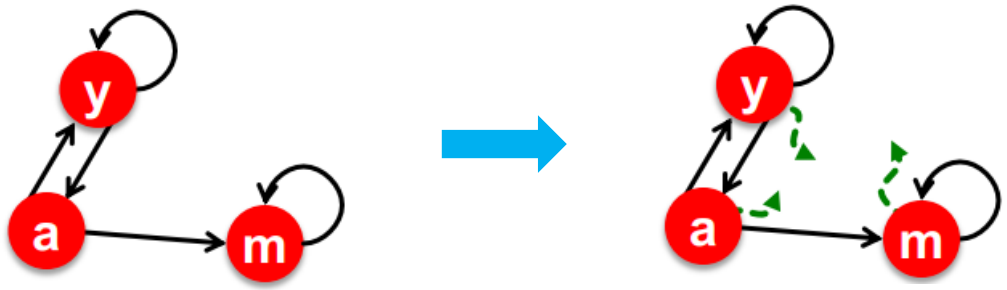
# PageRank: Solution to Spider Traps

Google solution for spider traps:

- At each time step, the random surfer has two options
  - With probability $\beta$, follow a link at random
  - With probability $1 - \beta$, jump to a random page
  - Common values for $\beta$ are in the range 0.8 to 0.9

Result:

- Surfer will teleport out of spider trap within a few time steps



# Summary Questions of the lecture

- Define the SCC in a graph and present a computation issue to find a SCC containing a node $v$.
- Discuss the conceptual picture of the Web graph obtained from the result in page 13 of lecture note 14.
- Explain the “Flow’ model for PageRank and discuss its validity on why it is worth.
- Present a random work interpretation of ‘power iteration’ method of eigenvector formulation for PageRank.