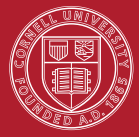


Cornell University

Lecture 3 - The algebra of networks' graphs

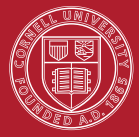
Graph-Based Data Science For Networked Systems
ECE 5260 – ORIE 5735

Anna Scaglione

Cornell Tech

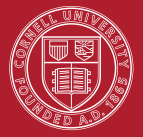
February 2, 2026

Content



Cornell University

Outline



Cornell University

Moving on a graph

- In some applications the data of interest can be modeled as the *sequence of nodes visited on a graph*
- These visits of nodes in many cases are modeled as a stochastic process called random walk
- Irrespective of the domain application how easy it is to navigate the entire graph is often a feature of interest to understand how efficient is the interaction in the network, since it affects
  - how graph nodal data can change in light of a triggering event that propagates through the network
  - how well goods, money, data, opinions, viruses flow and spread in an network.

Moving on a graph

- In some applications the data of interest can be modeled as the *sequence of nodes visited on a graph*
- These visits of nodes in many cases are modeled as a stochastic process called random walk
- Irrespective of the domain application how easy it is to navigate the entire graph is often a feature of interest to understand how efficient is the interaction in the network, since it affects
  - how graph nodal data can change in light of a triggering event that propagates through the network
  - how well goods, money, data, opinions, viruses flow and spread in an network.

Moving on a graph

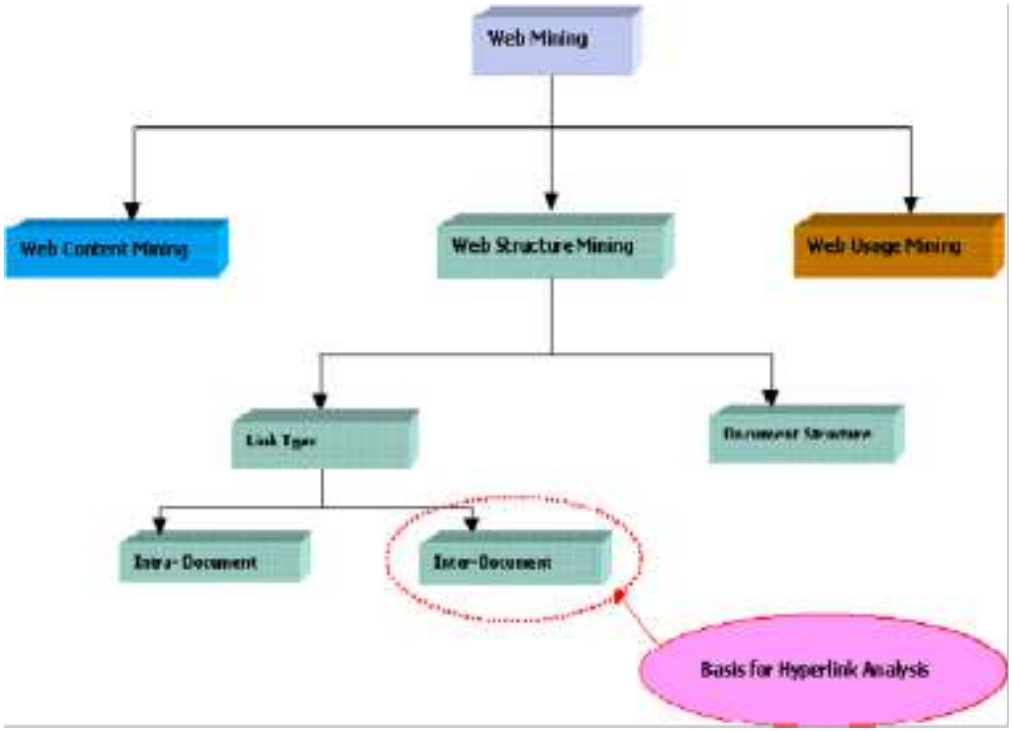
- In some applications the data of interest can be modeled as the *sequence of nodes visited on a graph*
- These visits of nodes in many cases are modeled as a stochastic process called random walk
- Irrespective of the domain application how easy it is to navigate the entire graph is often a feature of interest to understand how efficient is the interaction in the network, since it affects
  - how graph nodal data can change in light of a triggering event that propagates through the network
  - how well goods, money, data, opinions, viruses flow and spread in an network.

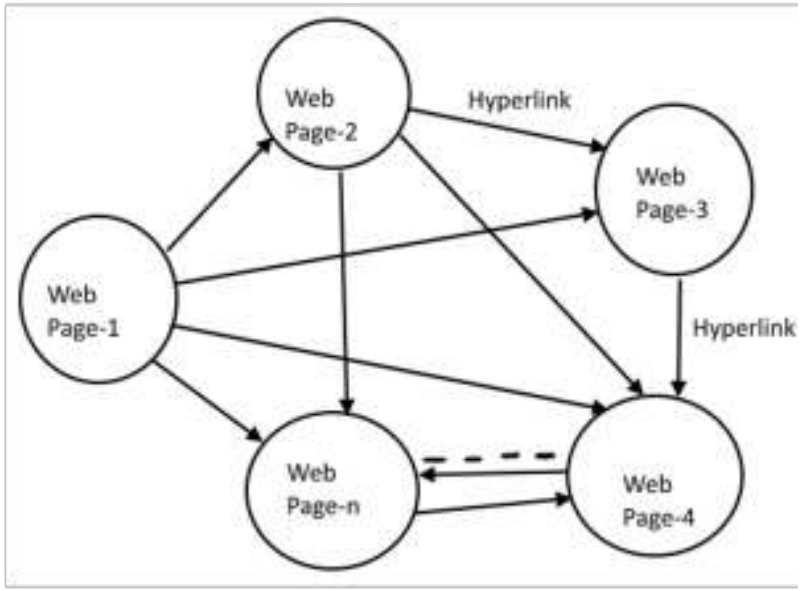
Moving on a graph

- In some applications the data of interest can be modeled as the *sequence of nodes visited on a graph*
- These visits of nodes in many cases are modeled as a stochastic process called random walk
- Irrespective of the domain application how easy it is to navigate the entire graph is often a feature of interest to understand how efficient is the interaction in the network, since it affects
  - how graph nodal data can change in light of a triggering event that propagates through the network
  - how well goods, money, data, opinions, viruses flow and spread in an network.

Example: browsing trends

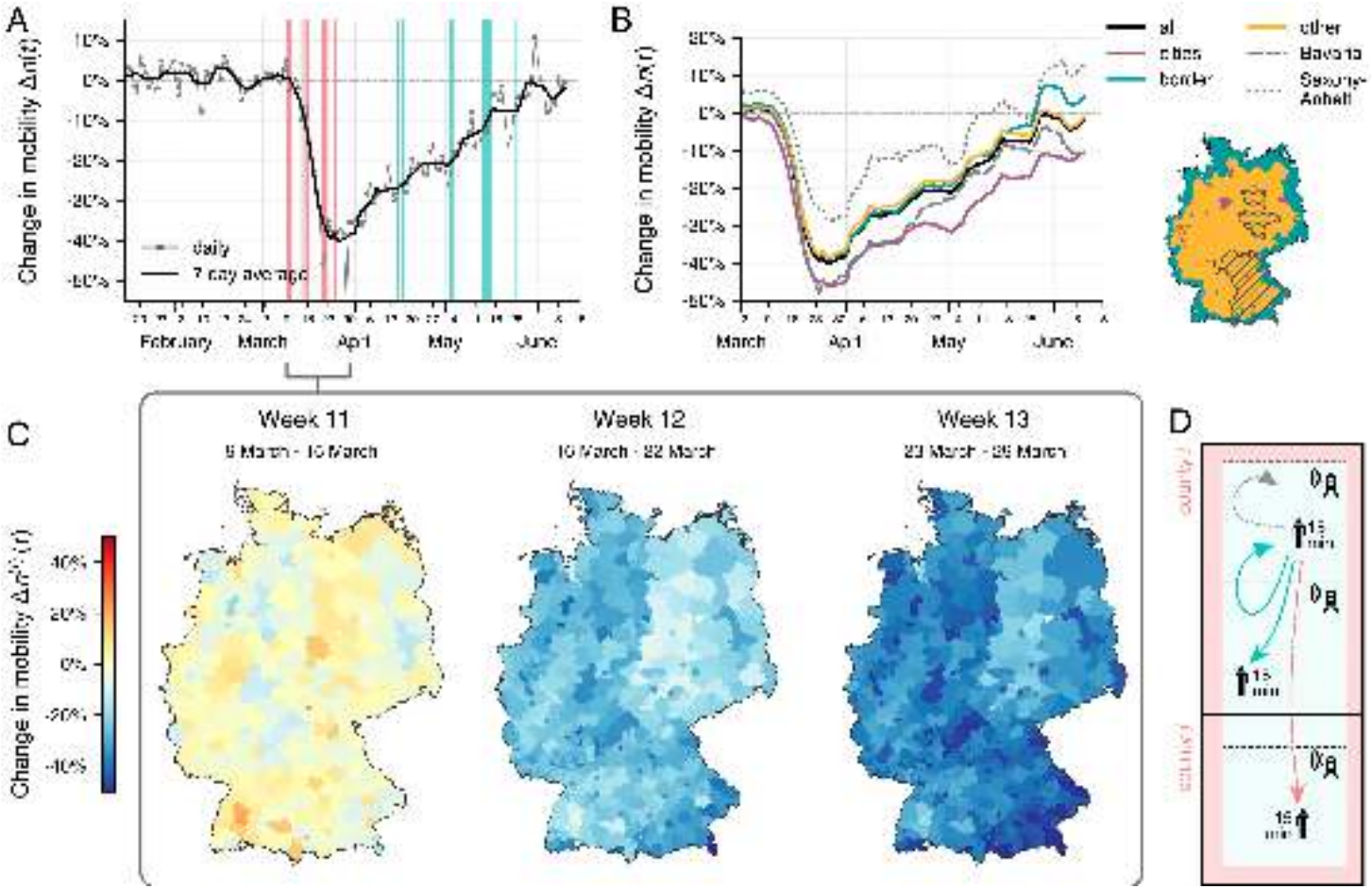
- For example, in webometrics the classic example is the users browsing hystory is the pattern in which users navigate hyperlinks (a *fingerprint* for the user)





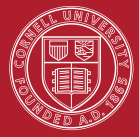
Example: epidemics

- This is an example where the dynamics of the spread of an state (virus infection, idea, failure) are affected by navigating the human network



Moving on a graph: walk

Graphs topological characteristics



Walk

A walk is a sequence alternating vertexes and the edges that connect them and ending with a node.

$$\mathfrak{w}_{v_0,v_n} = \{v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n\}.$$

The length of the walk is n, i.e. the number of edges in $\mathfrak{w}$.

Open walk

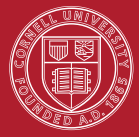
A walk is said to be an open walk if the starting and ending vertices are different i.e. the origin vertex and terminal vertex are different.

Closed walk

A walk is said to be a closed walk if the starting and ending vertices are identical.

Moving on a graph: walk

Graphs topological characteristics



Walk

A walk is a sequence alternating vertexes and the edges that connect them and ending with a node.

$$\mathfrak{w}_{v_0,v_n} = \{v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n\}.$$

The length of the walk is n, i.e. the number of edges in $\mathfrak{w}$.

Open walk

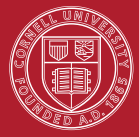
A walk is said to be an open walk if the starting and ending vertices are different i.e. the origin vertex and terminal vertex are different.

Closed walk

A walk is said to be a closed walk if the starting and ending vertices are identical.

Moving on a graph: walk

Graphs topological characteristics



Walk

A walk is a sequence alternating vertexes and the edges that connect them and ending with a node.

$$\mathfrak{w}_{v_0,v_n} = \{v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n\}.$$

The length of the walk is n, i.e. the number of edges in $\mathfrak{w}$.

Open walk

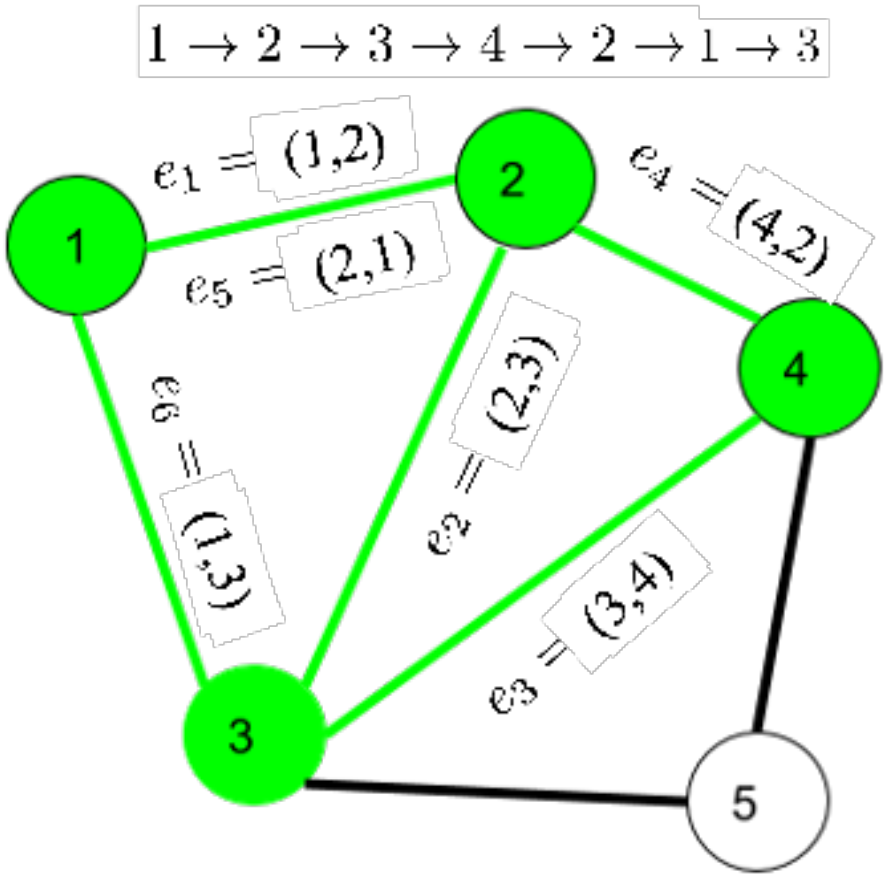
A walk is said to be an open walk if the starting and ending vertices are different i.e. the origin vertex and terminal vertex are different.

Closed walk

A walk is said to be a closed walk if the starting and ending vertices are identical.

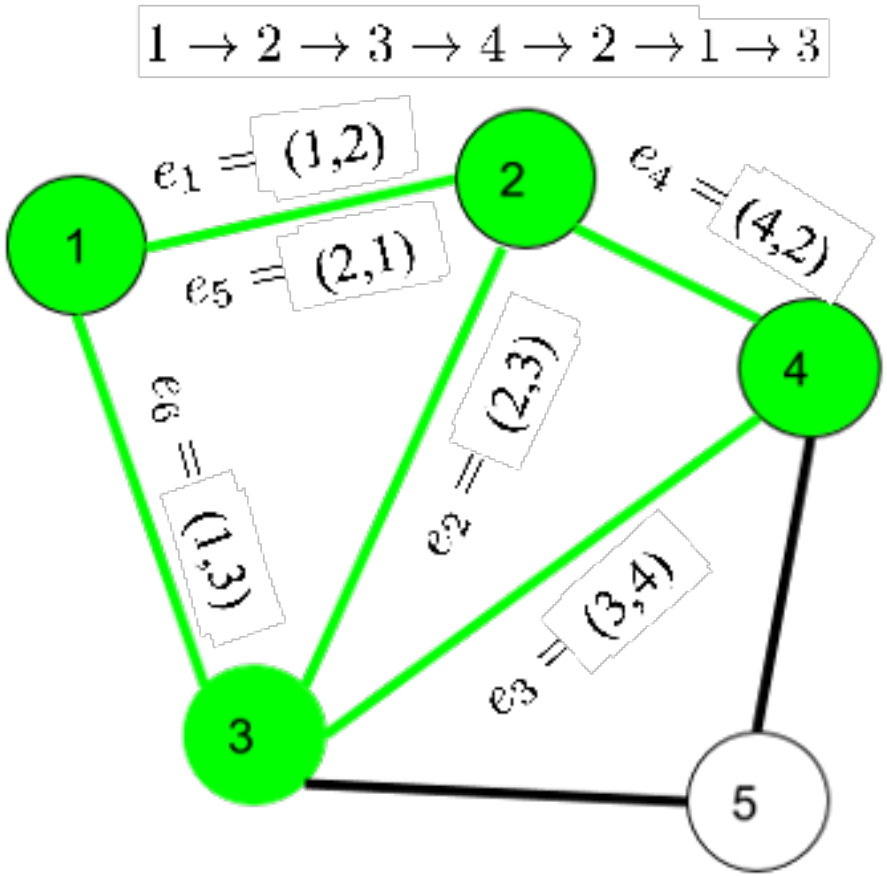
Example of walk

- The nodes are visited multiple times in general.
- $1 \to 2 \to 3 \to 4 \to 2 \to 1 \to 3$ (shown in the figure) is a walk (more specifically an open walk)
- The first five steps of the walk are $1 \to 2 \to 3 \to 4 \to 2 \to 1$, which is a closed walk.



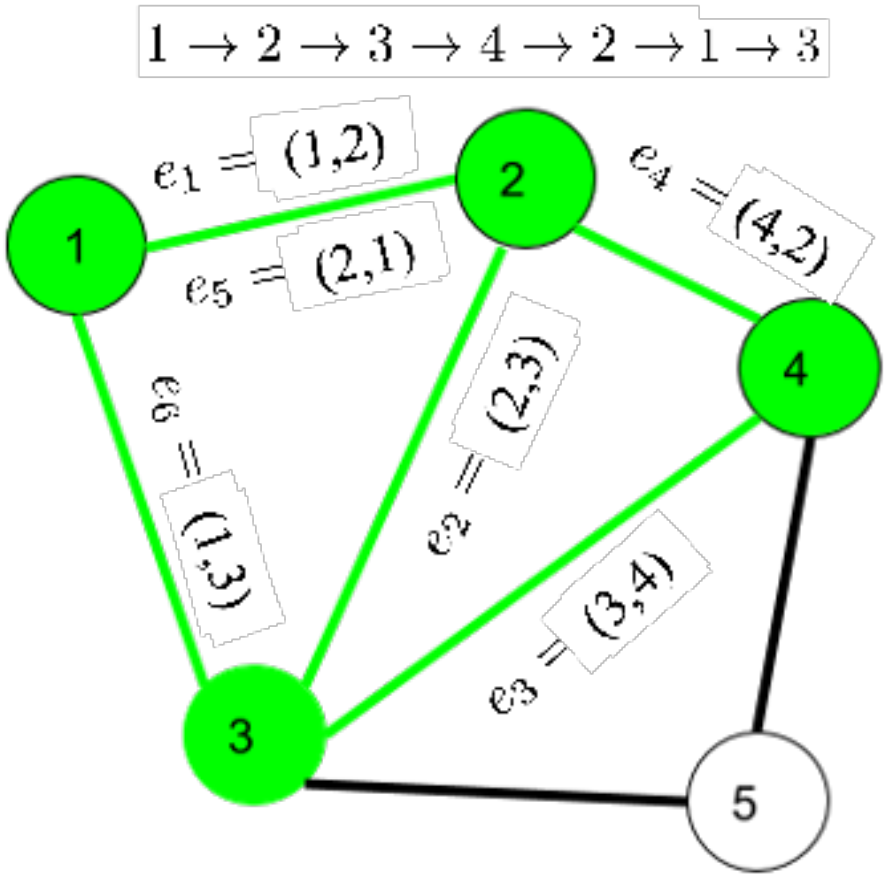
Example of walk

- The nodes are visited multiple times in general.
- $1 \to 2 \to 3 \to 4 \to 2 \to 1 \to 3$ (shown in the figure) is a walk (more specifically an open walk)
- The first five steps of the walk are $1 \to 2 \to 3 \to 4 \to 2 \to 1$, which is a closed walk.



Example of walk

- The nodes are visited multiple times in general.
- $1 \to 2 \to 3 \to 4 \to 2 \to 1 \to 3$ (shown in the figure) is a walk (more specifically an open walk)
- The first five steps of the walk are $1 \to 2 \to 3 \to 4 \to 2 \to 1$, which is a closed walk.



Trail and circuit

Graphs topological characteristics

Trail

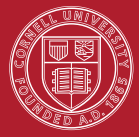
A **trail** between two nodes $\mathfrak{t}_{u,v}$ is a walk without repeated edges.

Circuit

A **trail** is a closed trail. Can have repeated vertices only. Note that nodes can be visited multiple times in a trail.

Trail and circuit

Graphs topological characteristics



Trail

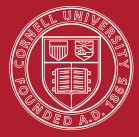
A **trail** between two nodes $\mathfrak{t}_{u,v}$ is a walk without repeated edges.

Circuit

A **trail** is a closed trail. Can have repeated vertices only. Note that nodes can be visited multiple times in a trail.

Trail and circuit

Graphs topological characteristics



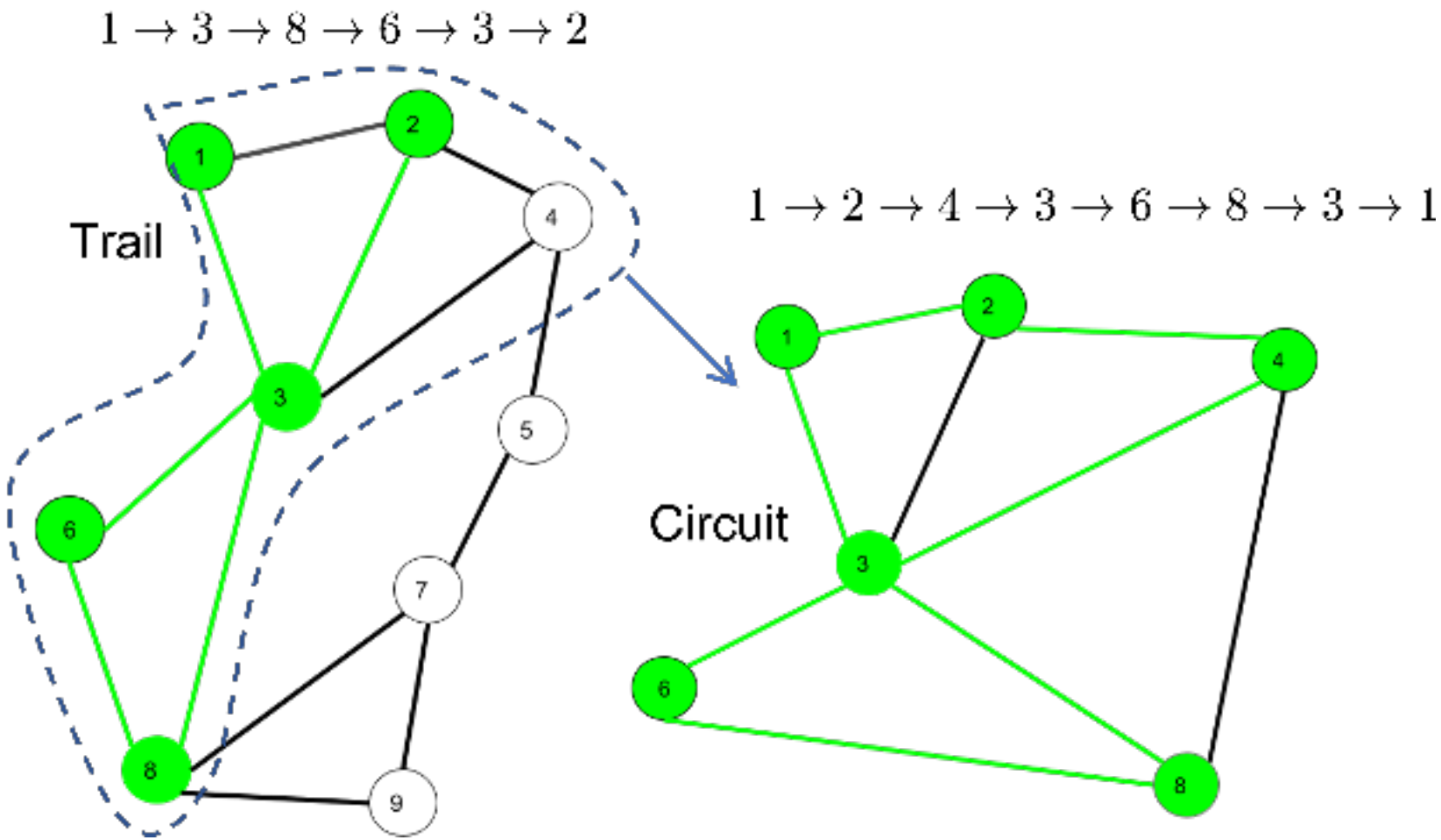
Trail

A **trail** between two nodes $\mathfrak{t}_{u,v}$ is a walk without repeated edges.

Circuit

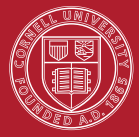
A **trail** is a closed trail. Can have repeated vertices only. Note that nodes can be visited multiple times in a trail.

Examples of trail and circuit



Path and cycle

Graphs topological characteristics



Path

A **path** between two nodes $\mathfrak{p}_{u,v}$ is a trail $\mathfrak{t}_{u,v}$ without repeated nodes.

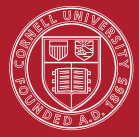
Cycle

A **cycle** is a non-empty trail in which only the first and last vertices are equal and the edges are different too.

- For trees, forests and poly trees any walk that starts at the root or an internal nodes and ends to a leaf is a path.

Path and cycle

Graphs topological characteristics



Path

A **path** between two nodes $\mathfrak{p}_{u,v}$ is a trail $\mathfrak{t}_{u,v}$ without repeated nodes.

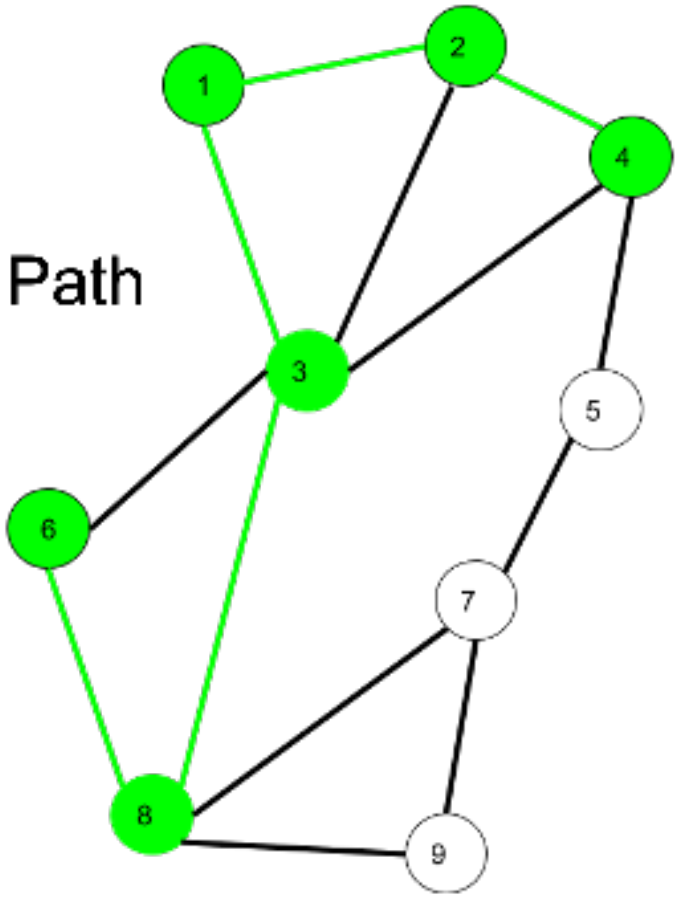
Cycle

A **cycle** is a non-empty trail in which only the first and last vertices are equal and the edges are different too.

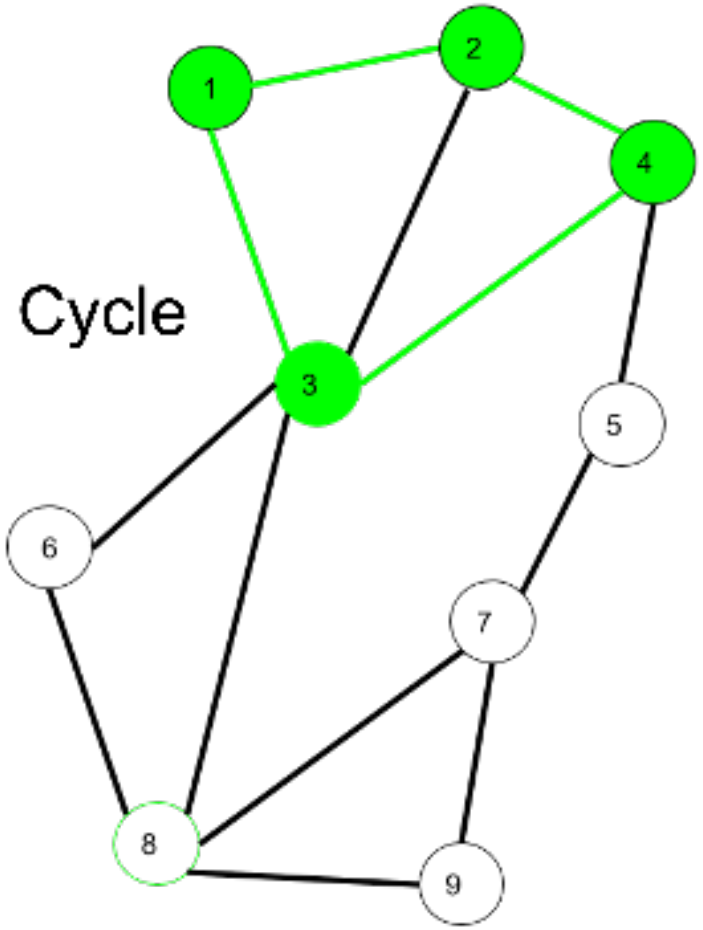
- For trees, forests and poly trees any walk that starts at the root or an internal nodes and ends to a leaf is a path.

Examples of path and cycle

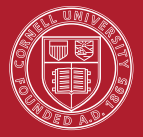
$6 \to 8 \to 3 \to 1 \to 2 \to 4$



$1 \to 2 \to 4 \to 3 \to 1$



Outline

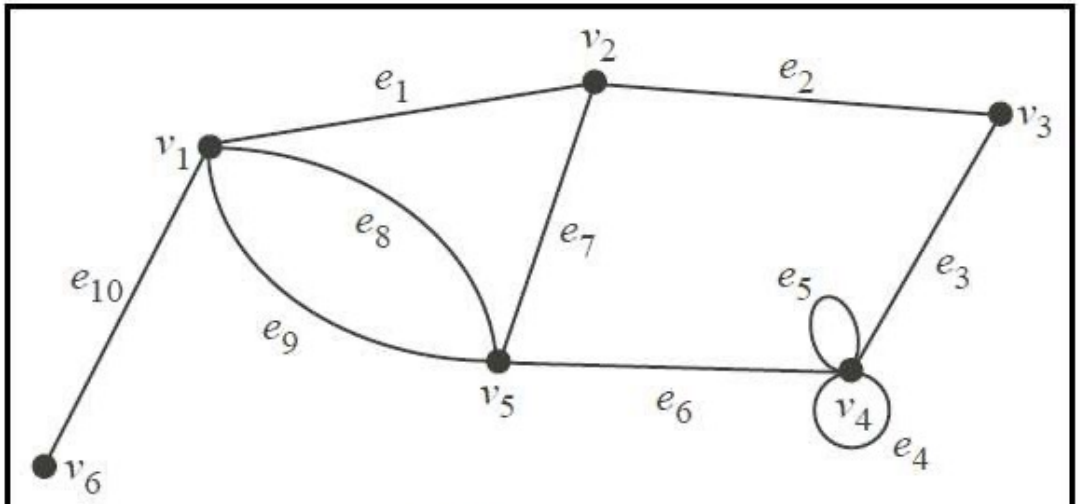


Cornell University

Distance on a graph

Graphs topological characteristics

- The graph distance of two nodes is the length of the shortest path that connects them.
- The path distance for a weighted graph is $\sum_{e \in \mathfrak{p}_{uv}} w_e$ is the edge weight).
- The graph diameter is the longest shortest path bewteen any node pair.



Graph G

Open Walk: $v_2, e_7, v_5, e_8, v_1, e_8, v_5, e_6, v_4, e_5, v_4, e_5, v_4$

Closed Walk: $v_4, e_5, v_4, e_3, v_3, e_2, v_2, e_7, v_5, e_6, v_4$

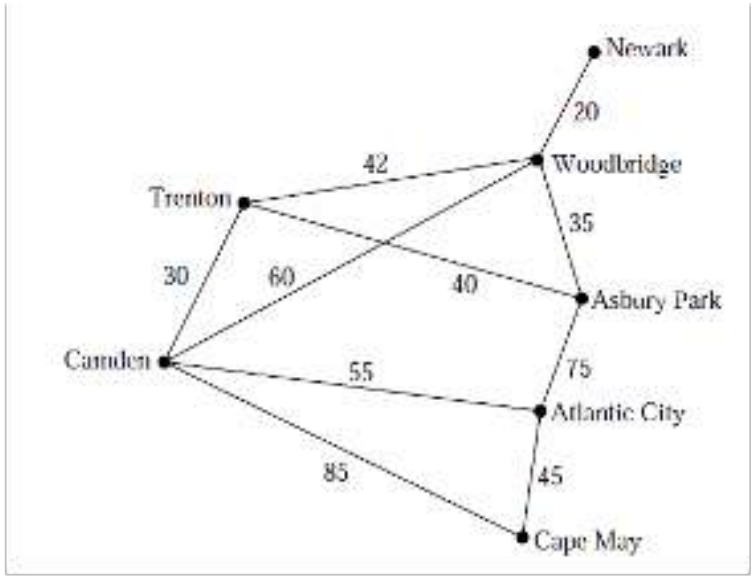
Trail: $v_1, e_8, v_5, e_9, v_1, e_1, v_2, e_7, v_5, e_6, v_4, e_5, v_4, e_4, v_4$

Path: $v_2, e_7, v_5, e_6, v_4, e_3, v_3$

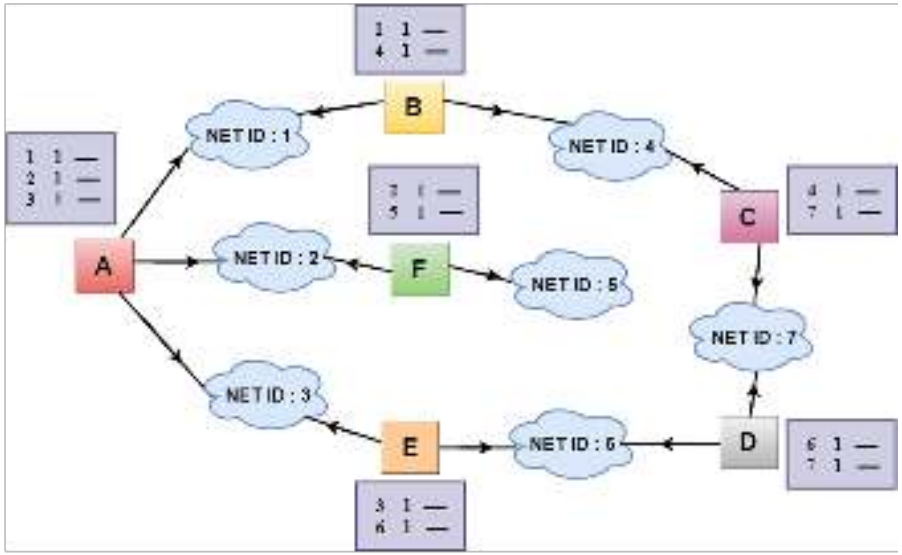
Circuit: $v_2, e_7, v_5, e_6, v_4, e_3, v_3, e_2, v_2$

Why is this notion important?

- One of the most important features of a graph model is how difficult it is to reach its node from a certain origin
- It is important for flow networks, like routing or transportation



Transportation network

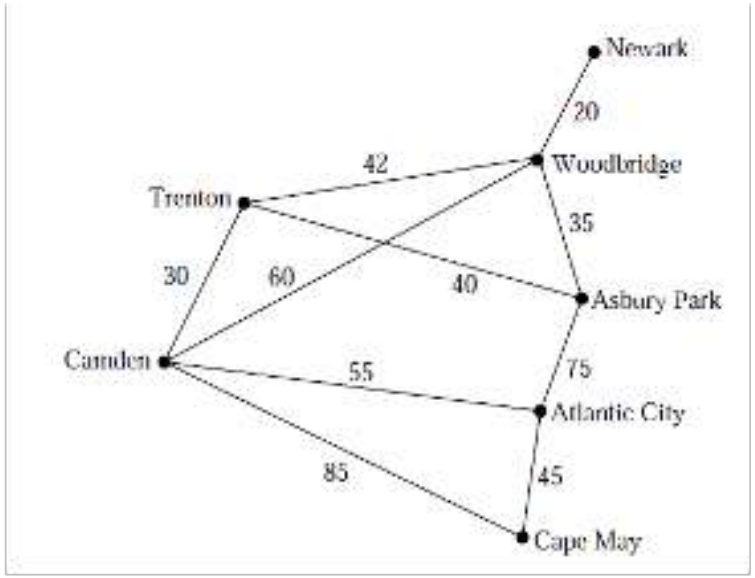


Communication network

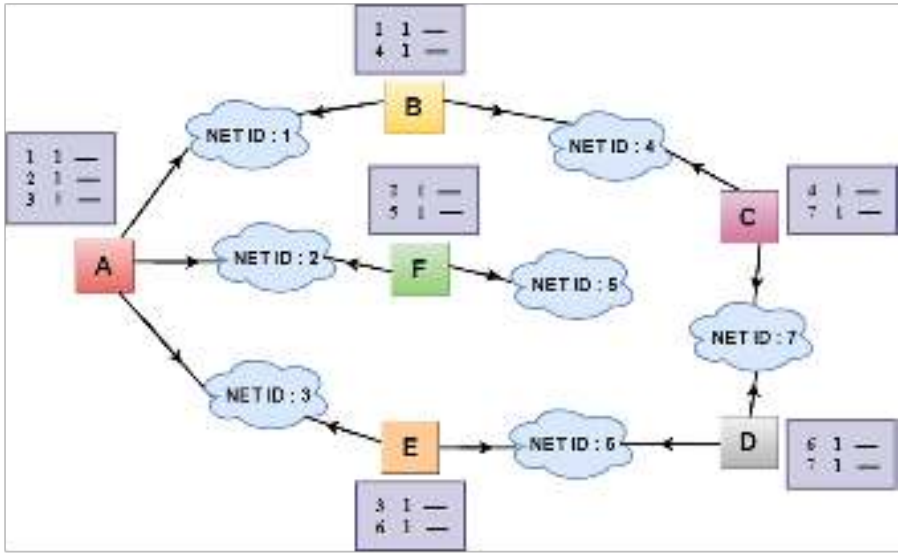
- It is a feature that helps understanding how well connected the network is and correlates with fast graph dynamics.

Why is this notion important?

- One of the most important features of a graph model is how difficult it is to reach its node from a certain origin
- It is important for flow networks, like routing or transportation



Transportation network

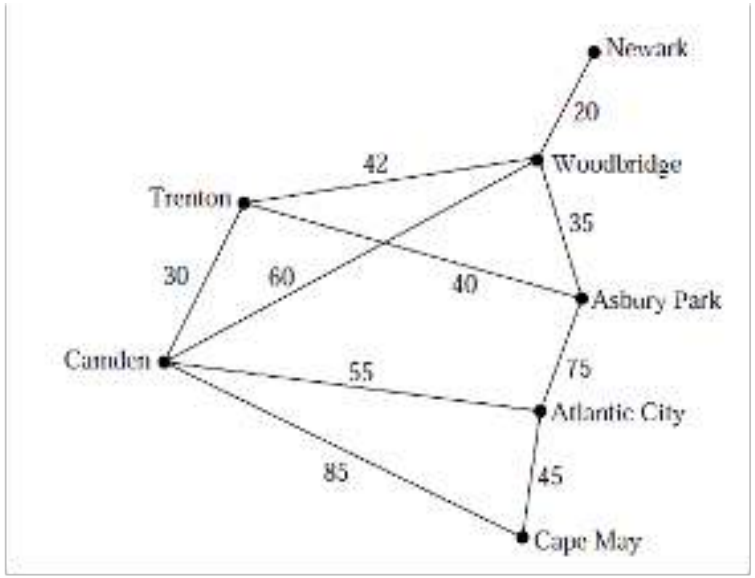


Communication network

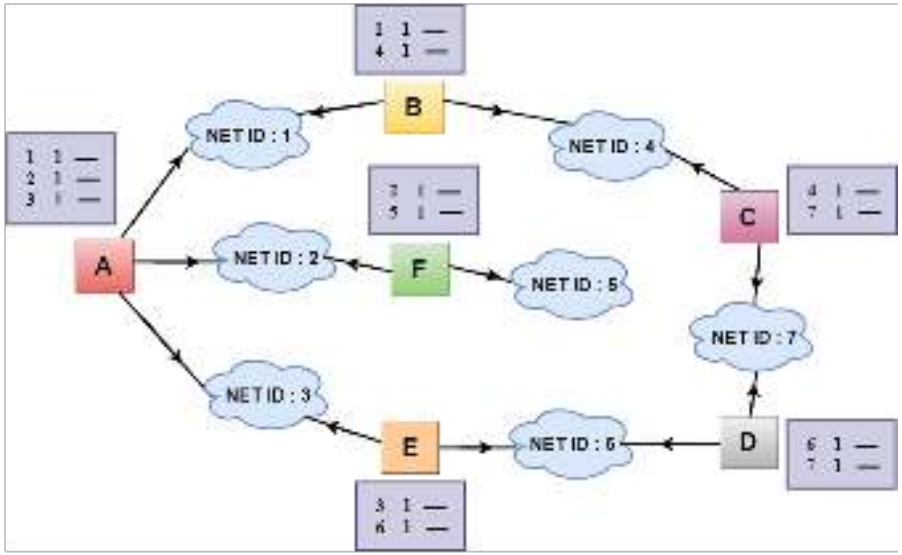
- It is a feature that helps understanding how well connected the network is and correlates with fast graph dynamics.

Why is this notion important?

- One of the most important features of a graph model is how difficult it is to reach its node from a certain origin
- It is important for flow networks, like routing or transportation



Transportation network

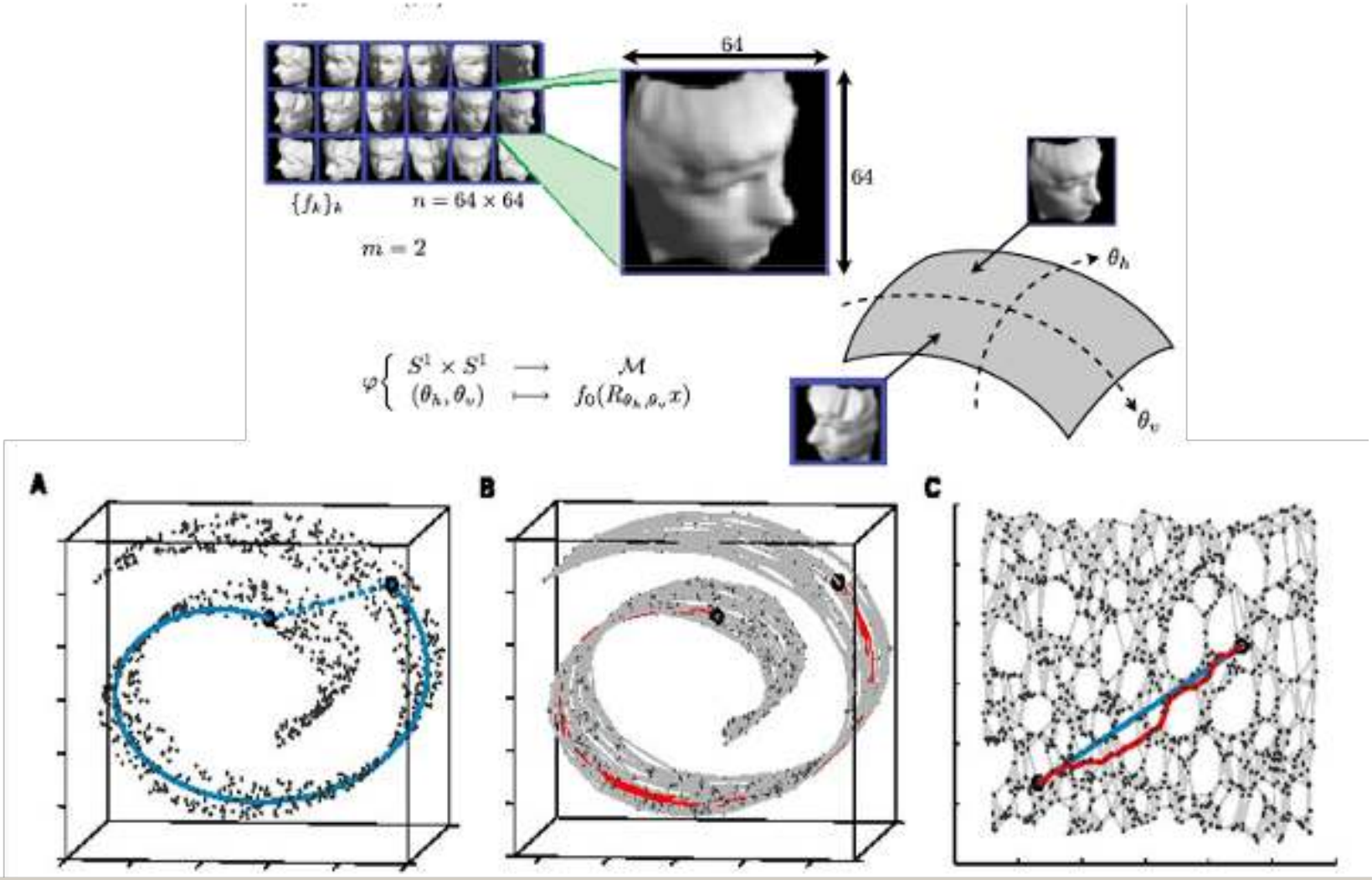


Communication network

- It is a feature that helps understanding how well connected the network is and correlates with fast graph dynamics.

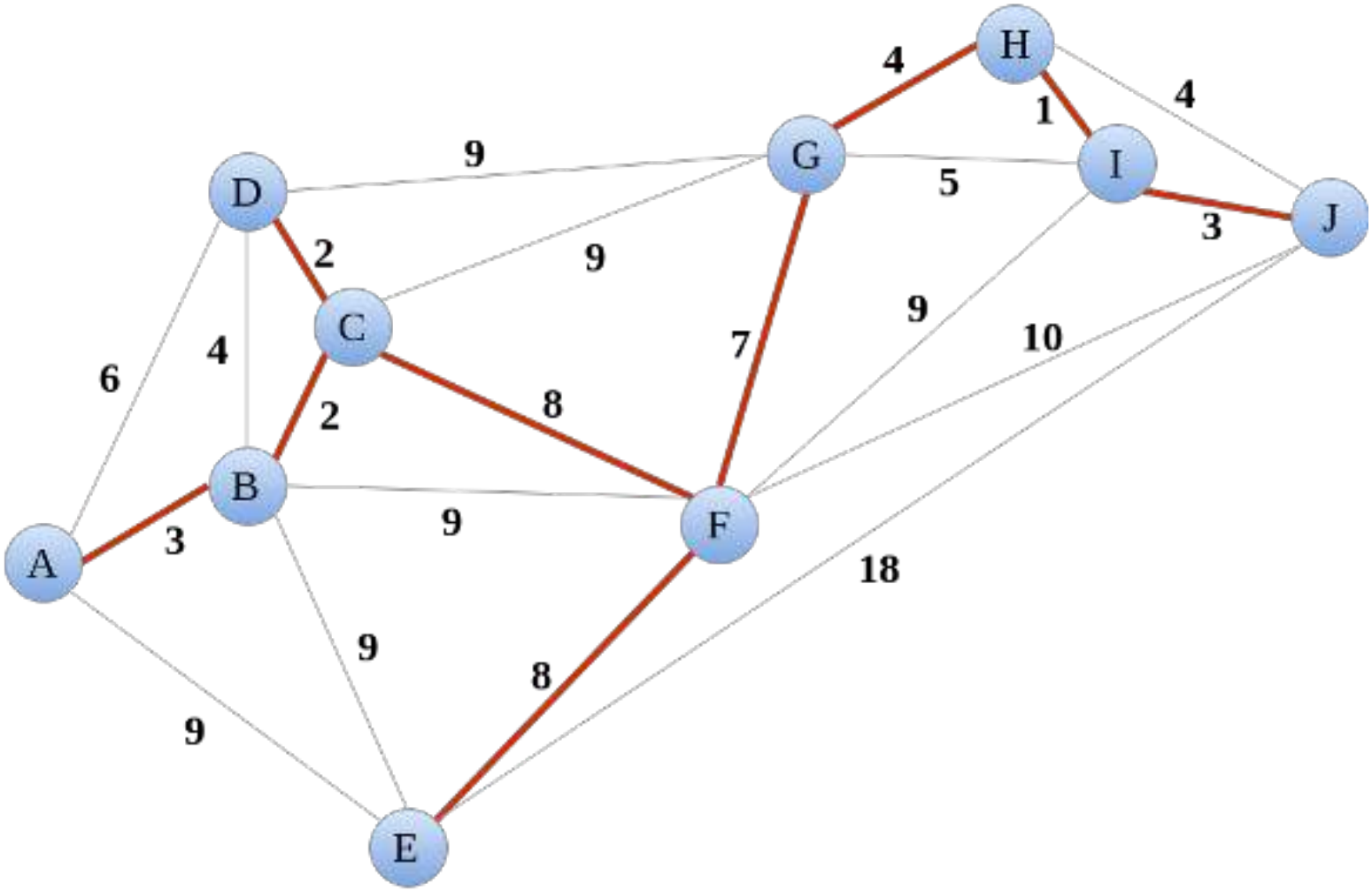
Distance in graph representing data

- We mentioned that graph can be a proxy to understand the low dimensional structure of data



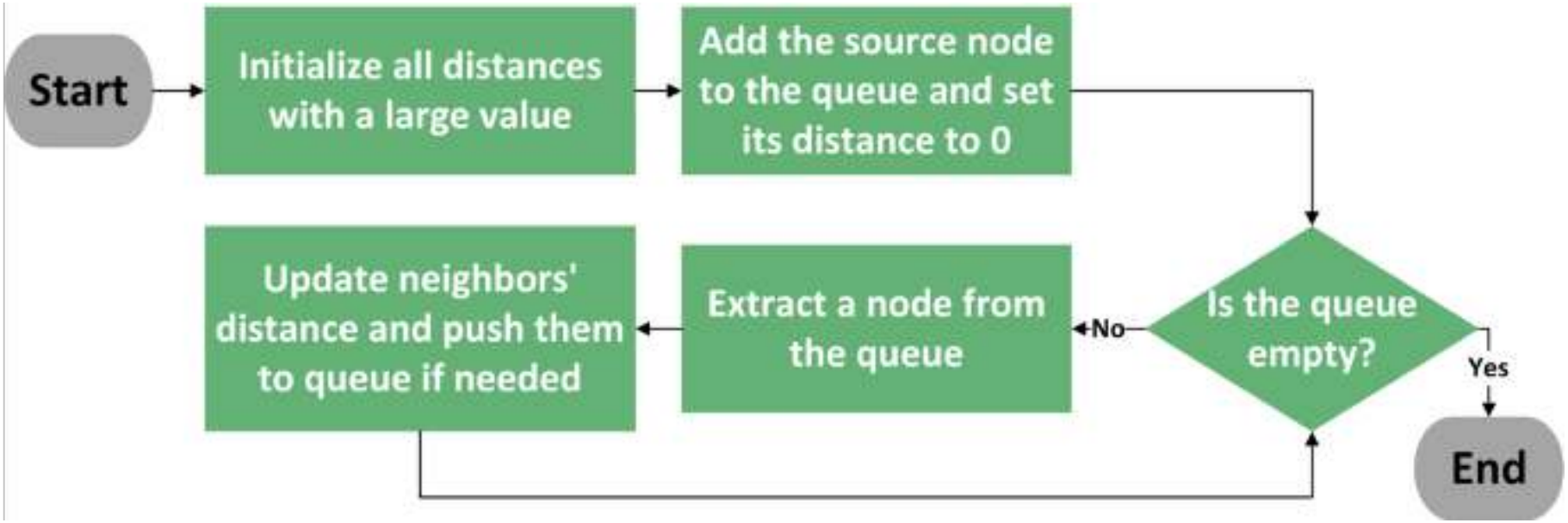
Spanning Trees

- A spanning tree connects all the vertices of a graph with the minimum possible number of edges. It is not unique in general.

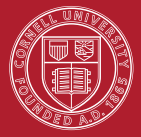


Spanning Trees

- Breadth First Search or Dijkstra's algorithms find it with complexity $\Theta(|\mathcal{E}| + |\mathcal{V}|)$ or $\Theta(|\mathcal{E}| \log(|\mathcal{V}|) + |\mathcal{V}|)$. Dijkstra's alg. can minimize the sum of edge weights not just path lengths.



Networkx functions



In Networkx you can use the functions:

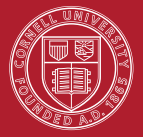
- MINIMUM_SPANNING_TREE(G) that uses Kruskal's algorithm to find a spanning tree from a certain source node.
- SHORTEST_PATH(G) which has the option of using Dijkstra or Bellman-Ford algorithms. It can use the weights, interpreted as edge distances/costs.
- There are several related functions to calculate functions of the shortest path (the existence of a path, all paths, the length of the shortest path, the average length etc.)

Networkx functions

In Networkx you can use the functions:

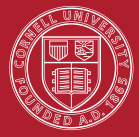
- MINIMUM_SPANNING_TREE(G) that uses Kruskal's algorithm to find a spanning tree from a certain source node.
- SHORTEST_PATH(G) which has the option of using Dijkstra or Bellman-Ford algorithms. It can use the weights, interpreted as edge distances/costs.
- There are several related functions to calculate functions of the shortest path (the existence of a path, all paths, the length of the shortest path, the average length etc.)

Walks vs paths



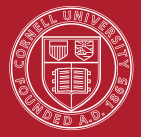
- We will see that walks are easy to count algebraically using the adjacency matrix
- Paths are combinatorially hard (NP-hard variants appear quickly)
- This is why shortest paths calculations are algorithmic, not spectral

Outline



Cornell University

Outline

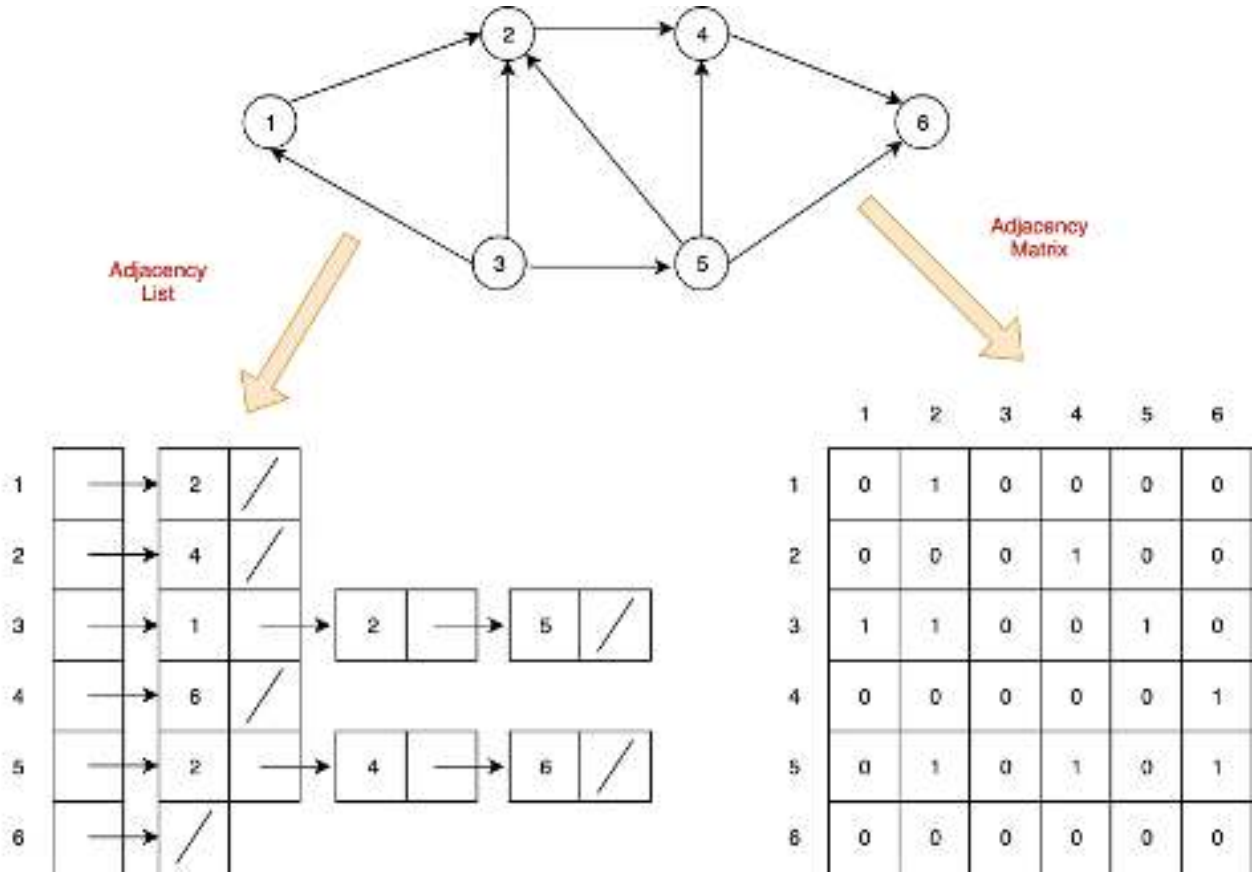


Cornell University

Premise
Algebra of networks

Cornell University

Matrices are not a compact representation of a graph when it is sparse. The adjacency list requires $O(|\mathcal{V}| + |\mathcal{E}|)$ and the matrix $O(|\mathcal{V}|^2)$.



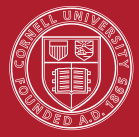
Premise
Algebra of networks

- They are, however, very useful to understand how to model data that are explained by network dynamics, and inform algorithms to process them.
- They are also useful to explore the topology: Counting walks, measuring reachability, detecting communities, and modeling dynamics all reduce to matrix operations.
- Perhaps one of the most interesting is the Laplacian, which is used for various types of analyses (the study is called spectral graph theory
- The Laplacian also emerges in a number of network dynamics and is fundamental in Graph Signal Processing.

Premise
Algebra of networks

- They are, however, very useful to understand how to model data that are explained by network dynamics, and inform algorithms to process them.
- They are also useful to explore the topology: Counting walks, measuring reachability, detecting communities, and modeling dynamics all reduce to matrix operations.
- Perhaps one of the most interesting is the Laplacian, which is used for various types of analyses (the study is called spectral graph theory
- The Laplacian also emerges in a number of network dynamics and is fundamental in Graph Signal Processing.

Premise
Algebra of networks

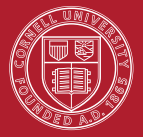


- They are, however, very useful to understand how to model data that are explained by network dynamics, and inform algorithms to process them.
- They are also useful to explore the topology: Counting walks, measuring reachability, detecting communities, and modeling dynamics all reduce to matrix operations.
- Perhaps one of the most interesting is the Laplacian, which is used for various types of analyses (the study is called spectral graph theory
- The Laplacian also emerges in a number of network dynamics and is fundamental in Graph Signal Processing.

Premise
Algebra of networks

- They are, however, very useful to understand how to model data that are explained by network dynamics, and inform algorithms to process them.
- They are also useful to explore the topology: Counting walks, measuring reachability, detecting communities, and modeling dynamics all reduce to matrix operations.
- Perhaps one of the most interesting is the Laplacian, which is used for various types of analyses (the study is called spectral graph theory
- The Laplacian also emerges in a number of network dynamics and is fundamental in Graph Signal Processing.

Outline



Cornell University

Adjacency matrix

- The adjacency matrix $\boldsymbol{A}$ elements for a directed graph are:

$$A_{uv} = \begin{cases} 1 & \text{if there is an edge from } v \text{ to } u, \text{ i.e. } (v,u) \in \mathcal{E} \\ 0 & \text{else.} \end{cases}$$

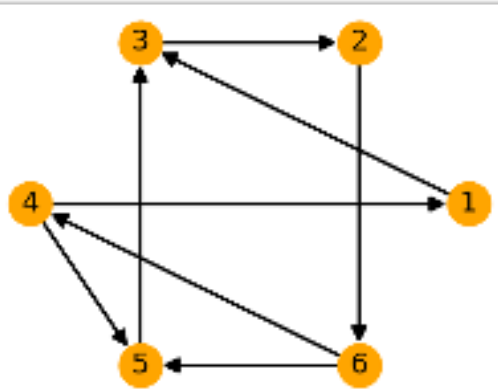
- The definition above accounts only for the topology. The weighted adjacency matrix has elements:

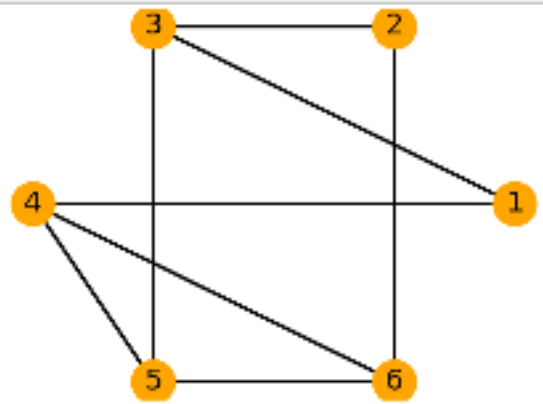
$$A_{uv} = \begin{cases} w_{uv} & \text{if there is an edge from } v \text{ to } u \text{ with weight } w_{uv} \\ 0 & \text{else.} \end{cases}.$$

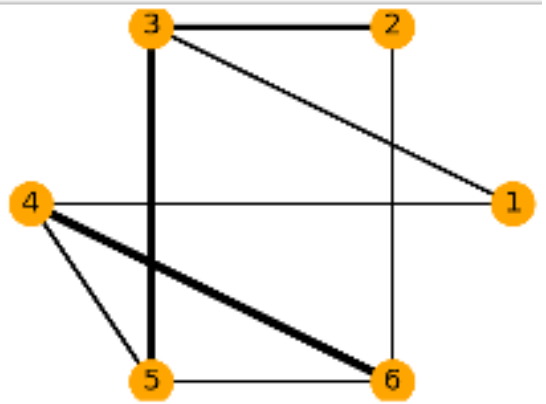
  The diagonal terms are zero if the network has no self-loops.
- For undirected we need to add $A_{uv} = A_{vu}$, i.e. $\boldsymbol{A} = \boldsymbol{A}^\top$.
- Note: In networkx the adjacency generated is equal to $\boldsymbol{A}^\top$.

Visualization of different types of graphs

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0. & 0. & 4.5 & 3.1 & 0. & 0. \\ 0. & 0. & 7.25 & 0. & 0. & 2.1 \\ 4.5 & 7.25 & 0. & 0. & 8.5 & 0. \\ 3.1 & 0. & 0. & 0. & 4.75 & 10.1 \\ 0. & 0. & 8.5 & 4.75 & 0. & 3.25 \\ 0. & 2.1 & 0. & 10.1 & 3.25 & 0. \end{bmatrix}$$







- On the left, a directed graph - note $\boldsymbol{A} \neq \boldsymbol{A}^\top$
- Middle and right side, undirected unweighted and weighted graphs

Degrees from the adjacency matrix

- Let $\mathbf{1}$ be the all ones vector. Note that for an unweighted and undirected graph/adjacency matrix the vector of degrees is $\boldsymbol{d} = \boldsymbol{A}\mathbf{1}$.
- For an unweighted directed graph
  - $\boldsymbol{d}^+ = \boldsymbol{A}\mathbf{1}$ is the vector of in-degrees $[\boldsymbol{d}^+]_u = |\mathcal{N}_u^+|$.
  - $\boldsymbol{d}^- = \mathbf{1}^\top \boldsymbol{A}$ is the vector of out-degrees $[\boldsymbol{d}^-]_u = |\mathcal{N}_u^-|$.
- Assuming that $\boldsymbol{A}$ is the weighted adjacency matrix, instead of the degrees the elements of $\boldsymbol{A}\mathbf{1}$ are the sums of the weights: $\sum_{v \in \mathcal{N}_u} w_{uv}$

Degrees from the adjacency matrix

- Let $\mathbf{1}$ be the all ones vector. Note that for an unweighted and undirected graph/adjacency matrix the vector of degrees is $\boldsymbol{d} = \boldsymbol{A}\mathbf{1}$.
- For an unweighted directed graph
  - $\boldsymbol{d}^+ = \boldsymbol{A}\mathbf{1}$ is the vector of in-degrees $[\boldsymbol{d}^+]_u = |\mathcal{N}_u^+|$.
  - $\boldsymbol{d}^- = \mathbf{1}^\top \boldsymbol{A}$ is the vector of out-degrees $[\boldsymbol{d}^-]_u = |\mathcal{N}_u^-|$.
- Assuming that $\boldsymbol{A}$ is the weighted adjacency matrix, instead of the degrees the elements of $\boldsymbol{A}\mathbf{1}$ are the sums of the weights: $\sum_{v \in \mathcal{N}_u} w_{uv}$

Degrees from the adjacency matrix

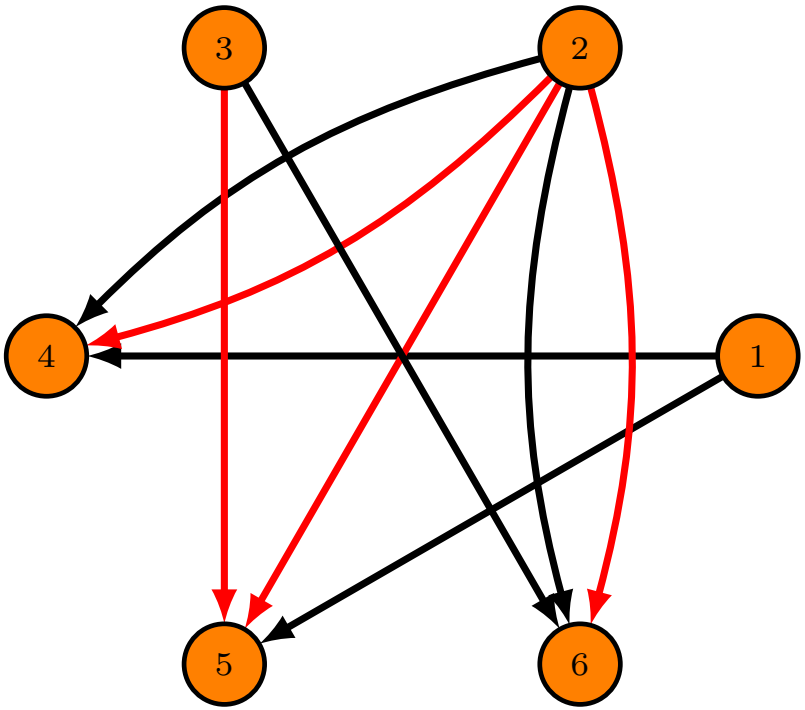
- Let $\mathbf{1}$ be the all ones vector. Note that for an unweighted and undirected graph/adjacency matrix the vector of degrees is $\boldsymbol{d} = \boldsymbol{A}\mathbf{1}$.
- For an unweighted directed graph
  - $\boldsymbol{d}^+ = \boldsymbol{A}\mathbf{1}$ is the vector of in-degrees $[\boldsymbol{d}^+]_u = |\mathcal{N}_u^+|$.
  - $\boldsymbol{d}^- = \mathbf{1}^\top \boldsymbol{A}$ is the vector of out-degrees $[\boldsymbol{d}^-]_u = |\mathcal{N}_u^-|$.
- Assuming that $\boldsymbol{A}$ is the weighted adjacency matrix, instead of the degrees the elements of $\boldsymbol{A}\mathbf{1}$ are the sums of the weights: $\sum_{v \in \mathcal{N}_u} w_{uv}$

Adjacency matrix for multigraphs

- It is often better to look at the multi-graphs as a series of graphs $\mathcal{G}(t) = (\mathcal{V}, \mathcal{E}(t))$, but the convention used to define their adjacency matrix of a multi-graph is setting

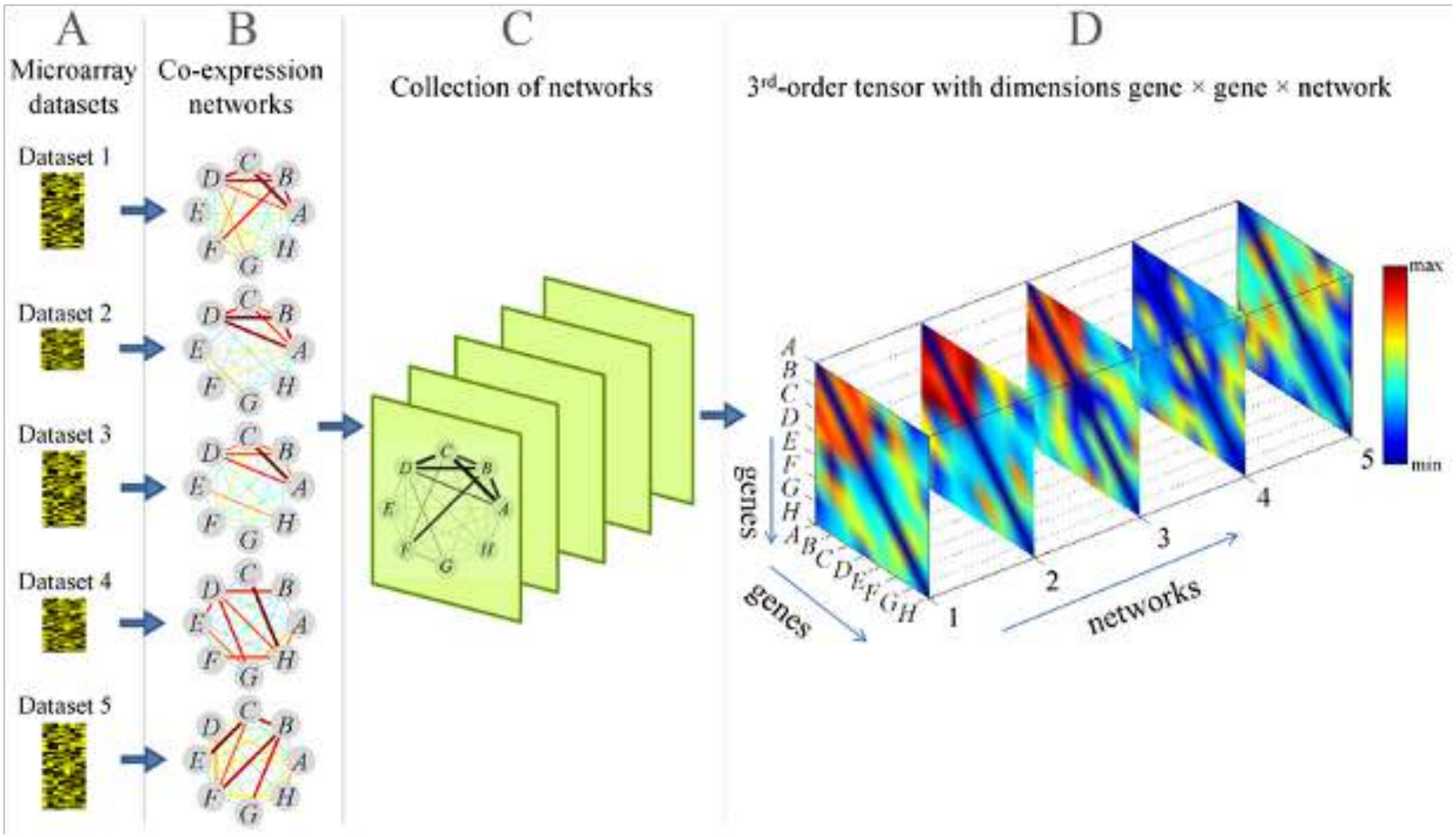
$$A_{uv} = \text{number of edges from } v \text{ to } u \text{ in the set } \bigcup_{t=1}^{T} \mathcal{E}(t) = \sum_{t=1}^{T} A_{uv}(t).$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 & 0 \end{bmatrix}$$



Multigraphs adjacency tensor representation

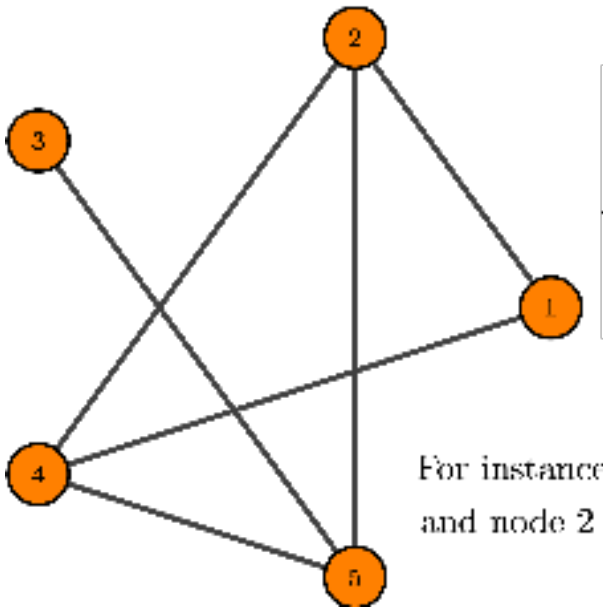
A preferable representation is to consider a tensor $A_{uvt} = A_{uv}(t)$, whose *slices* are the matrices that correspond to the different graphs adjacency matrices, i.e. $\boldsymbol{A}_t : [\boldsymbol{A}_t]_{uv} = A_{uv}(t)$



Number of walks

Number of walks

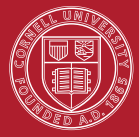
Consider a directed or undirected graph and a positive integer k. The number of directed and undirected walks respectively from node i to node j of length k is the entry on row i and column j of the matrix $\boldsymbol{A}^k$, where $\boldsymbol{A}$ is the adjacency matrix.



$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}, \quad \boldsymbol{A}^2 = \begin{bmatrix} 2 & 2 & 1 & 1 & 1 \\ 2 & 3 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 4 & 2 \\ 1 & 1 & 2 & 2 & 3 \end{bmatrix}, \quad \boldsymbol{A}^3 = \begin{bmatrix} 2 & 3 & 3 & 6 & 5 \\ 3 & 4 & 5 & 7 & 7 \\ 3 & 5 & 2 & 6 & 3 \\ 6 & 7 & 6 & 6 & 7 \\ 5 & 7 & 3 & 7 & 4 \end{bmatrix}$$

For instance the element $[\boldsymbol{A}^3]_{1,2} = 5$ is the number of walks between node 1 and node 2 of length 3. These walks are: $1 \to 4 \to 5 \to 2$, $1 \to 4 \to 1 \to 2$, $1 \to 2 \to 1 \to 2$, $1 \to 2 \to 4 \to 2$, $1 \to 2 \to 5 \to 2$

Connected graphs



Strongly connected graphs

If a graph is strongly connected then there exist a finite k, $1 \leq k \leq +\infty$ such that $\sum_{i=0}^{k} \boldsymbol{A}^i$ has all elements that are non-zero. For fully connected, clearly $k = 1$.

- In fact the elements of $\boldsymbol{A}^k$ are. equal to the number of walks from any two nodes
- This is way larger than the number of paths

Graph Laplacian

- Another very important matrix for the analysis of both topological as well as graph data processes is Laplacian matrix. It is defined for undirected graphs as:

$$\boldsymbol{L} = \text{diag}(\boldsymbol{d}) - \boldsymbol{A}$$

- We will study this operator in detail. It measures how different are a node value from its neighbors

$$[\boldsymbol{L}\boldsymbol{x}]_i = \sum_{j \in \mathcal{N}_i} (x_i - x_j)$$

- We will see that if the Laplacian is never full rank. In fact, $\boldsymbol{L}\mathbf{1} = \mathbf{0}$. An undirected graph is strongly connected if the rank of $\boldsymbol{L}$ is $N - 1$.
- If the Laplacian has rank $N - k$ then there are k strongly connected components.
- These are results of *algebraic graph theory*.

Graph Laplacian

- Another very important matrix for the analysis of both topological as well as graph data processes is Laplacian matrix. It is defined for undirected graphs as:

$$\boldsymbol{L} = \text{diag}(\boldsymbol{d}) - \boldsymbol{A}$$

- We will study this operator in detail. It measures how different are a node value from its neighbors

$$[\boldsymbol{L}\boldsymbol{x}]_i = \sum_{j \in \mathcal{N}_i} (x_i - x_j)$$

- We will see that if the Laplacian is never full rank. In fact, $\boldsymbol{L}\mathbf{1} = \mathbf{0}$. An undirected graph is strongly connected if the rank of $\boldsymbol{L}$ is $N - 1$.
- If the Laplacian has rank $N - k$ then there are k strongly connected components.
- These are results of *algebraic graph theory*.

Graph Laplacian

- Another very important matrix for the analysis of both topological as well as graph data processes is Laplacian matrix. It is defined for undirected graphs as:

$$\boldsymbol{L} = \text{diag}(\boldsymbol{d}) - \boldsymbol{A}$$

- We will study this operator in detail. It measures how different are a node value from its neighbors

$$[\boldsymbol{L}\boldsymbol{x}]_i = \sum_{j \in \mathcal{N}_i} (x_i - x_j)$$

- We will see that if the Laplacian is never full rank. In fact, $\boldsymbol{L}\boldsymbol{1} = \boldsymbol{0}$. An undirected graph is strongly connected if the rank of $\boldsymbol{L}$ is $N - 1$.
- If the Laplacian has rank $N - k$ then there are k strongly connected components.
- These are results of *algebraic graph theory*.

Number of spanning trees

- There are two results on **the number of spanning trees** $t(\mathcal{G})$:

Cayley's formula

The number of spanning trees for the complete graph is n^{n-2}.

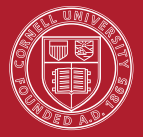
Kirchoff theorem

Let $\boldsymbol{L} = \text{diag}(\boldsymbol{d}) - \boldsymbol{A}$, where $\text{diag}(\boldsymbol{v})$ is a diagonal matrix with diagonal elements equal to v_i and $\boldsymbol{d}$ is the degree sequence. Then, denote by $\lambda_1, \lambda_2, \ldots, \lambda_N$ are the eigenvalues of the matrix $\boldsymbol{L}$:

$$t(\mathcal{G}) = \frac{1}{|\mathcal{V}|}\lambda_1 \cdot \lambda_2 \ldots \cdot \lambda_{N-1}$$

The Laplacian reveals many interesting properties for the corresponding graph and appears in many graph data models.

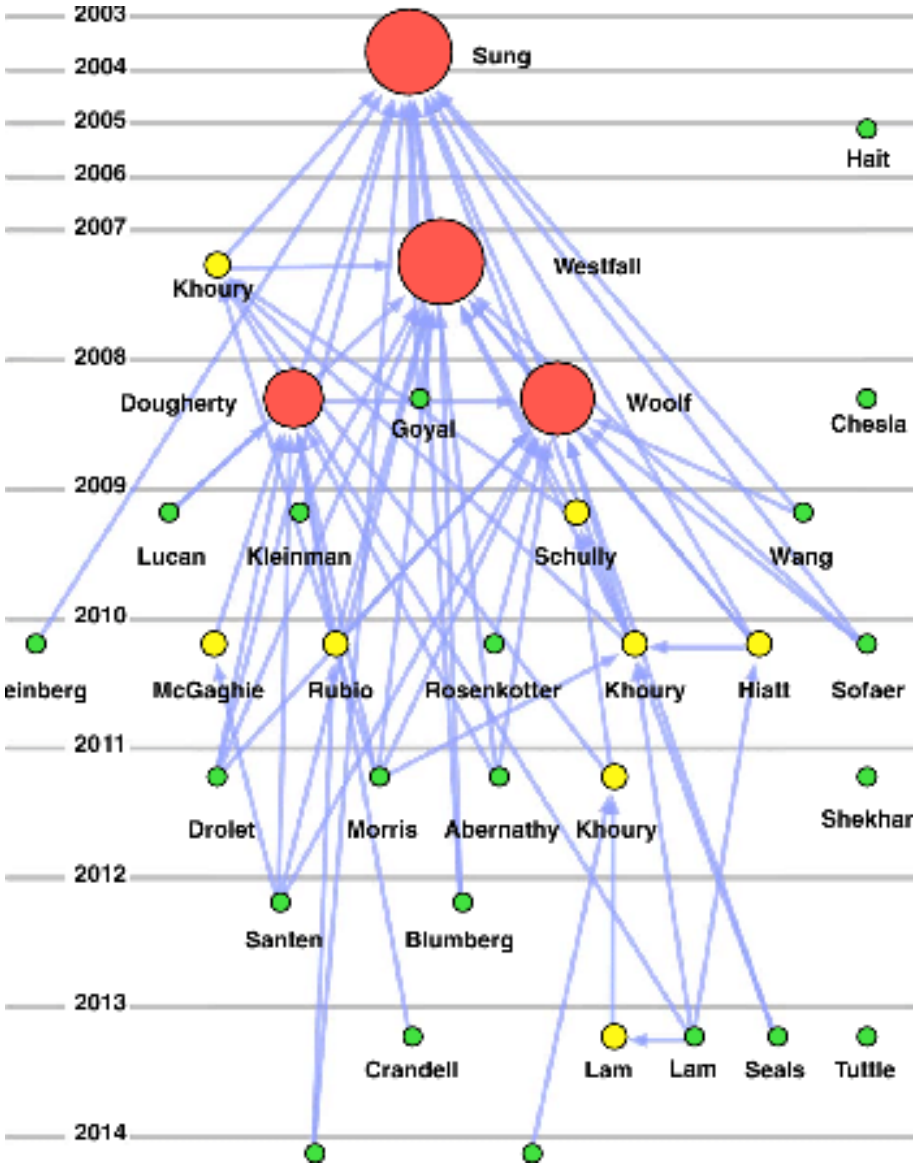
Outline



Cornell University

Directed Acyclic Graphs (DAG): Example

- Often, if the nodes are properly ordered, the matrix $\boldsymbol{A}$ reveals an underlying fact about the graph. DAG example: citation graphs.



Directed Acyclic Graphs (DAG)

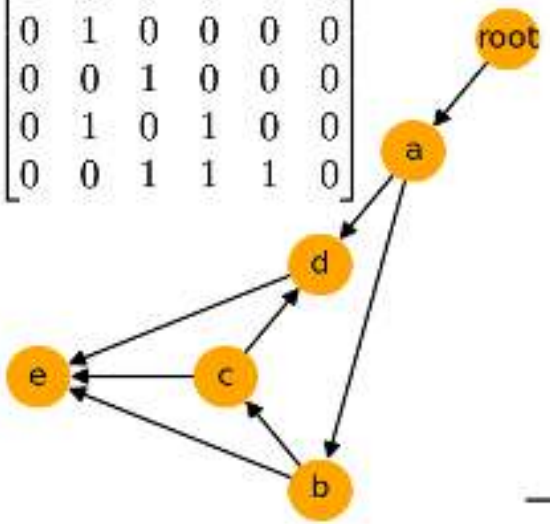
- One of such cases is that of DAGs, for which under proper ordering called *topological*, $\boldsymbol{A}$ appears as an upper triangular matrix.

Directed Acyclic Graph

Topological ordering

$\mathcal{V} = \{root,\ a,\ b,\ c,\ d,\ e\}$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$



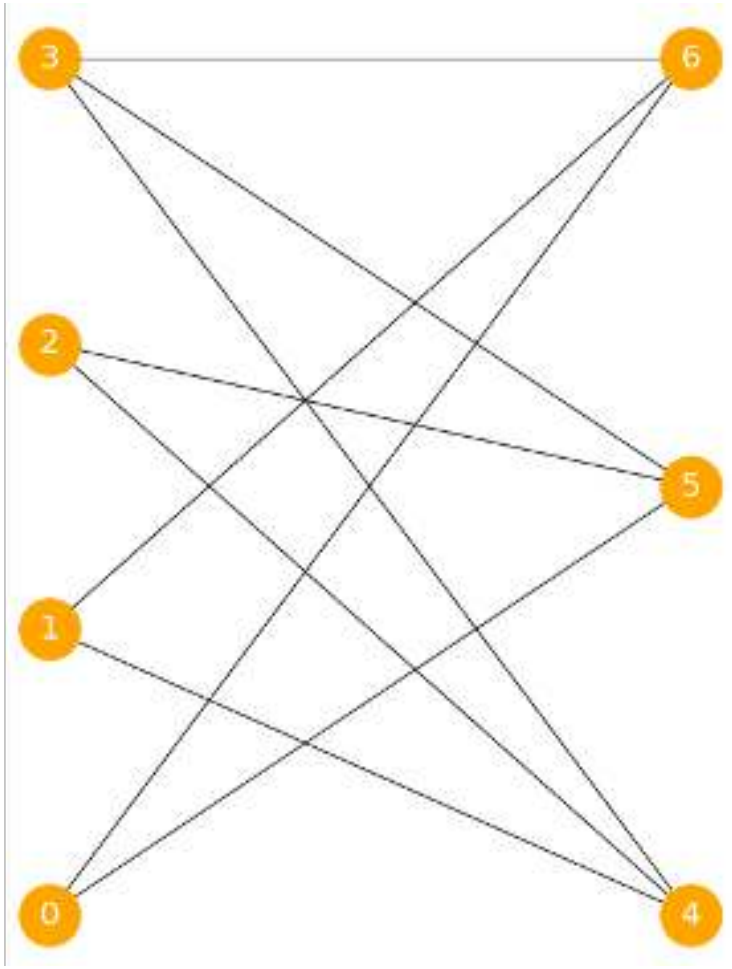
Algorithm 1: Kahn's Algorithm for Topological Sort

Data: A DAG G
Result: A topological sort of all vertices in G
Compute in-degree (number of incoming edges) for each vertex in G;
Put all the vertices with 0 in-degree into a queue Q;
Create an empty vertex list L;
while Q *is not empty* **do**
  Remove a vertex u from Q;
  Add u to the end of the L;
  foreach u*'s neighboring node* v **do**
    Decrease v's in-degree by 1;
    if v*'s in-degree is* 0 **then**
      Add v to Q;
    end
  end
end
return L;

Bipartite Graphs

- The adjacency matrix of bipartite graphs also has a natural order by placing the two groups together.

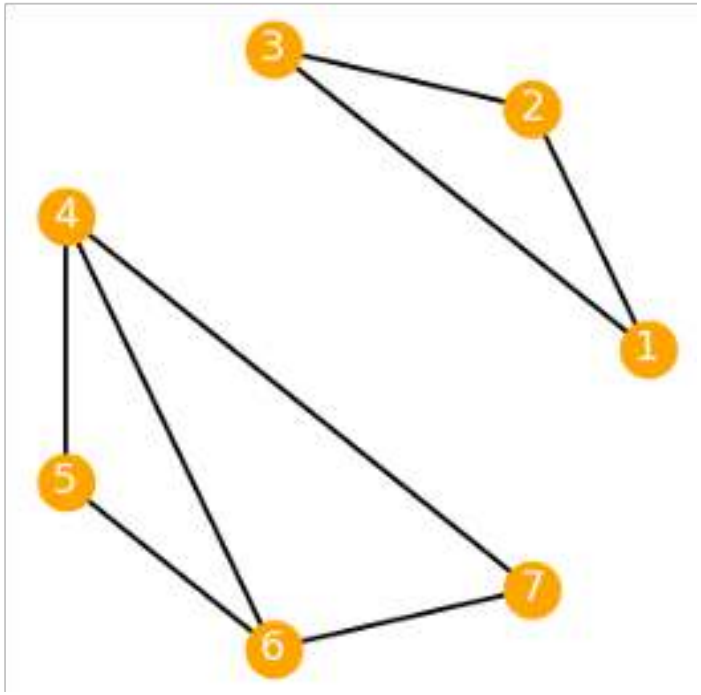
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$



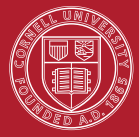
Disconnected Graphs

- The adjacency of a graph with disconnected component can also reveal the two communities. This graph is $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2$

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

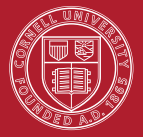


Outline



Cornell University

Concluding remarks



- We explored the properties of the adjacency matrix (completed in the next lecture)
- In the next lecture we will study the incidence matrix and its application to network flow problems.