CS 4984: Connectivity Matrices and Node Degrees

T. M. Murali

February 6, 2018
Definition of an Undirected Graph

- **Weighted, undirected graph** $G = (V, E, w)$:
  - set $V$ of nodes.
  - set $E$ of edges.
    - Each element of $E$ is an unordered pair of nodes.
    - Exactly one edge between any pair of nodes ($G$ is not a multigraph).
    - $G$ contains no self loops, i.e., edges of the form $(u, u)$.
  - Each edge $(u, v)$ in $E$ has a weight $w(u, v) \in \mathbb{R}$
    - Weight of each edge is usually positive.
    - $G$ is *unweighted* if all edges have weight 1.
Definition of an Undirected Graph

- **Weighted, undirected graph** $G = (V, E, w)$:
  - set $V$ of nodes.
  - set $E$ of edges.
    - Each element of $E$ is an unordered pair of nodes.
    - Exactly one edge between any pair of nodes ($G$ is not a multigraph).
    - $G$ contains no self loops, i.e., edges of the form $(u, u)$.
  - Each edge $(u, v)$ in $E$ has a weight $w(u, v) \in \mathbb{R}$
    - Weight of each edge is usually positive.
    - $G$ is **unweighted** if all edges have weight 1.
Definition of an Undirected Graph

**Weighted, undirected graph** $G = (V, E, w)$:

- set $V$ of nodes.
- set $E$ of edges.
  - Each element of $E$ is an unordered pair of nodes.
  - Exactly one edge between any pair of nodes ($G$ is not a multigraph).
  - $G$ contains no self loops, i.e., edges of the form $(u, u)$.
- Each edge $(u, v)$ in $E$ has a weight $w(u, v) \in \mathbb{R}$
  - Weight of each edge is usually positive.
  - $G$ is *unweighted* if all edges have weight 1.
Definition of an Undirected Graph

- **Weighted, undirected graph** $G = (V, E, w)$:
  - set $V$ of nodes.
  - set $E$ of edges.
    - Each element of $E$ is an unordered pair of nodes.
    - Exactly one edge between any pair of nodes ($G$ is not a multigraph).
    - $G$ contains no self loops, i.e., edges of the form $(u, u)$.
  - Each edge $(u, v)$ in $E$ has a weight $w(u, v) \in \mathbb{R}$
    - Weight of each edge is usually positive.
    - $G$ is *unweighted* if all edges have weight 1.
Definition of a Directed Graph

- **Weighted, directed graph** \( G = (V, E, w) \):
  - set \( V \) of nodes.
  - set \( E \) of edges.
    - Each element of \( E \) is an ordered pair of nodes.
    - \( e = (u, v) \): \( u \) is the *tail* of the edge \( e \), \( v \) is its *head*; \( e \) is *directed from \( u \) to \( v \).*
    - A pair of nodes \( \{u, v\} \) may be connected by at most two directed edges: \( (u, v) \) and \( (v, u) \).
    - \( G \) contains no self loops.
  - Each edge \( (u, v) \) in \( E \) has a weight \( w(u, v) \in \mathbb{R} \)
    - Weight of each edge is usually positive.
    - \( G \) is *unweighted* if all edges have weight 1.
Definition of a Directed Graph

- **Weighted, directed graph** \( G = (V, E, w) \):
  - set \( V \) of nodes.
  - set \( E \) of edges.
    - Each element of \( E \) is an ordered pair of nodes.
    - \( e = (u, v) \): \( u \) is the *tail* of the edge \( e \), \( v \) is its *head*; \( e \) is *directed from \( u \) to \( v \).*
    - A pair of nodes \( \{u, v\} \) may be connected by at most two directed edges: \( (u, v) \) and \( (v, u) \).
    - \( G \) contains no self loops.
  - Each edge \( (u, v) \) in \( E \) has a weight \( w(u, v) \in \mathbb{R} \)
    - Weight of each edge is usually positive.
    - \( G \) is *unweighted* if all edges have weight 1.
Definition of a Directed Graph

- **Weighted, directed graph** $G = (V, E, w)$:
  - set $V$ of nodes.
  - set $E$ of edges.
    - Each element of $E$ is an ordered pair of nodes.
    - $e = (u, v)$: $u$ is the *tail* of the edge $e$, $v$ is its *head*; $e$ is directed from $u$ to $v$.
    - A pair of nodes $\{u, v\}$ may be connected by at most two directed edges: $(u, v)$ and $(v, u)$.
    - $G$ contains no self loops.
  - Each edge $(u, v)$ in $E$ has a weight $w(u, v) \in \mathbb{R}$
    - Weight of each edge is usually positive.
    - $G$ is *unweighted* if all edges have weight 1.
Definition of a Directed Graph

- **Weighted, directed graph** $G = (V, E, w)$:
  - set $V$ of nodes.
  - set $E$ of edges.
    - Each element of $E$ is an ordered pair of nodes.
    - $e = (u, v)$: $u$ is the *tail* of the edge $e$, $v$ is its *head*; $e$ is directed from $u$ to $v$.
    - A pair of nodes $\{u, v\}$ may be connected by at most two directed edges: $(u, v)$ and $(v, u)$.
    - $G$ contains no self loops.
  - Each edge $(u, v)$ in $E$ has a weight $w(u, v) \in \mathbb{R}$
    - Weight of each edge is usually positive.
    - $G$ is *unweighted* if all edges have weight 1.
<table>
<thead>
<tr>
<th>Representation</th>
<th>Hierholzer’s Again</th>
<th>Visualisation</th>
<th>Degrees</th>
</tr>
</thead>
</table>

### Types of Brain Graphs

<table>
<thead>
<tr>
<th>Scale</th>
<th>Structural connectivity</th>
<th>Functional connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microscale</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mesoscale</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Macroscale</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Microscale**
  - **Structural connectivity**
    - SEM, Tracking neurons
    - Directed, weighted
  - **Functional connectivity**
    - SEM, Tracking neurons
    - Directed, weighted

- **Mesoscale**
  - **Structural connectivity**
    - Invasive tract tracing
    - Directed, weighted
  - **Functional connectivity**
    - Did not discuss
    - Directed, weighted

- ** Macroscale**
  - **Structural connectivity**
    - Diffusion MRI, tractography
    - Undirected, weighted
    - Weighted, can be negative
    - Directed, weighted
  - **Functional connectivity**
    - fMRI, correlations
    - Weighted, can be negative
    - Directed, weighted

---

*February 6, 2018 CS 4984: Computing the Brain*
## Types of Brain Graphs

<table>
<thead>
<tr>
<th>Representation</th>
<th>Hierholzer’s Again</th>
<th>Visualisation</th>
<th>Degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Types of Brain Graphs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Structural connectivity</strong></td>
<td><strong>Functional connectivity</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Microscale</strong></td>
<td>SEM, Tracking neurons</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mesoscale</strong></td>
<td>Did not discuss</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Macroscale</strong></td>
<td>Diffusion MRI, tractography</td>
<td>fMRI, correlations</td>
<td></td>
</tr>
</tbody>
</table>

### Microscale
- **SEM, Tracking neurons**

### Mesoscale
- **Invasive tract tracing**

### Macroscale
- **Diffusion MRI, tractography**
- **fMRI, correlations**

[Diagram of segmented neurons and layout graph showing connectivity graph with labels a, b, c, d.]

**Soma:**
- Neuron ID,
- three-dimensional coordinates, type

**Axonal branch:**
- Neuron ID,
- three-dimensional coordinates, diameter

**Dendritic branch:**
- Neuron ID,
- three-dimensional coordinates, diameter

**Synaptic junction:**
- Pre- and postneuron ID,
- three-dimensional coordinates,
- number of vesicles
# Types of Brain Graphs

<table>
<thead>
<tr>
<th>Representation</th>
<th>Structural connectivity</th>
<th>Functional connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microscale</td>
<td>SEM, Tracking neurons</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Directed, weighted</td>
<td></td>
</tr>
<tr>
<td>Mesoscale</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Macroscale</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of brain graphs](image)
# Types of Brain Graphs

<table>
<thead>
<tr>
<th>Microscale</th>
<th>Structural connectivity</th>
<th>Functional connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SEM, Tracking neurons</td>
<td>Electrodes, correlations</td>
</tr>
<tr>
<td></td>
<td>Directed, weighted</td>
<td>Weighted, can be negative, can be directed</td>
</tr>
<tr>
<td>Mesoscale</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Macroscale</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Segmented neurons**

- **Soma:**
  - Neuron ID,
  - three-dimensional coordinates, type
- **Axonal branch:**
  - Neuron ID,
  - three-dimensional coordinates, diameter
- **Dendritic branch:**
  - Neuron ID,
  - three-dimensional coordinates, diameter
- **Synaptic junction:**
  - Pre- and postneuron ID,
  - three-dimensional coordinates, number of vesicles

**Layout graph**

**Connectivity graph**

---

T. M. Murali  
February 6, 2018  
CS 4984: Computing the Brain
# Types of Brain Graphs

<table>
<thead>
<tr>
<th>Representation</th>
<th>Hierholzer’s Again</th>
<th>Visualisation</th>
<th>Degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Types of Brain Graphs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Structural connectivity</strong></td>
<td><strong>Functional connectivity</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Microscale</strong></td>
<td>SEM, Tracking neurons</td>
<td>Electrodes, correlations</td>
<td>Directed, weighted</td>
</tr>
<tr>
<td></td>
<td>Directed, weighted</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mesoscale</strong></td>
<td>Invasive tract tracing</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Macroscale</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Types of Brain Graphs

<table>
<thead>
<tr>
<th>Representation</th>
<th>Hierholzer’s Again</th>
<th>Visualisation</th>
<th>Degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microscale</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Types of Brain Graphs</td>
<td>Structural connectivity</td>
<td>Functional connectivity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEM, Tracking neurons</td>
<td>Electrodes, correlations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Directed, weighted</td>
<td>Weighted, can be negative, can be directed</td>
<td></td>
</tr>
<tr>
<td>Mesoscale</td>
<td>Invasive tract tracing</td>
<td>Directed, weighted</td>
<td></td>
</tr>
<tr>
<td>Macroscale</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of brain graphs](image-url)
## Types of Brain Graphs

<table>
<thead>
<tr>
<th></th>
<th>Structural connectivity</th>
<th>Functional connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Microscale</strong></td>
<td>SEM, Tracking neurons</td>
<td>Electrodes, correlations</td>
</tr>
<tr>
<td></td>
<td>Directed, weighted</td>
<td>Weighted, can be negative, can be directed</td>
</tr>
<tr>
<td><strong>Mesoscale</strong></td>
<td>Invasive tract tracing</td>
<td>Did not discuss</td>
</tr>
<tr>
<td></td>
<td>Directed, weighted</td>
<td></td>
</tr>
<tr>
<td><strong>Macroscale</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Types of Brain Graphs

<table>
<thead>
<tr>
<th>Scale</th>
<th>Structural connectivity</th>
<th>Functional connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microscale</td>
<td>SEM, Tracking neurons</td>
<td>Electrodes, correlations</td>
</tr>
<tr>
<td></td>
<td>Directed, weighted</td>
<td>Weighted, can be negative, can be directed</td>
</tr>
<tr>
<td></td>
<td>Invasive tract tracing</td>
<td>Did not discuss</td>
</tr>
<tr>
<td>Mesoscale</td>
<td>Directed, weighted</td>
<td></td>
</tr>
<tr>
<td>Macroscale</td>
<td>Diffusion MRI, tractography</td>
<td></td>
</tr>
</tbody>
</table>
## Types of Brain Graphs

<table>
<thead>
<tr>
<th>Scale</th>
<th>Structural Connectivity</th>
<th>Functional Connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Microscale</strong></td>
<td>SEM, Tracking neurons, Directed, weighted</td>
<td>Electrodes, correlations, Weighted, can be negative, can be directed</td>
</tr>
<tr>
<td><strong>Mesoscale</strong></td>
<td>Invasive tract tracing, Directed, weighted</td>
<td>Did not discuss</td>
</tr>
<tr>
<td>** Macroscale**</td>
<td>Diffusion MRI, tractography, Undirected, weighted</td>
<td></td>
</tr>
</tbody>
</table>

![Brain Graph Image](image-url)
# Types of Brain Graphs

<table>
<thead>
<tr>
<th>Scale</th>
<th>Structural connectivity</th>
<th>Functional connectivity</th>
</tr>
</thead>
</table>
| **Microscale** | SEM, Tracking neurons  
Directed, weighted | Electrodes, correlations  
Weighted, can be negative, can be directed |
| **Mesoscale**  | Invasive tract tracing  
Directed, weighted | Did not discuss                                    |
| **Macroscale** | Diffusion MRI, tractography  
Undirected, weighted | fMRI, correlations                                |

- **SEM**: Scanning electron microscopy
- **fMRI**: Functional magnetic resonance imaging
- **corr**: Correlations
- **dir**: Directed
- **undir**: Undirected
- **wei**: Weighted
- **neg**: Negative
- **inv**: Invasive
## Types of Brain Graphs

<table>
<thead>
<tr>
<th>Representation</th>
<th>Structural connectivity</th>
<th>Functional connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Microscale</strong></td>
<td>SEM, Tracking neurons</td>
<td>Electrodes, correlations</td>
</tr>
<tr>
<td></td>
<td>Directed, weighted</td>
<td>Weighted, can be negative, can be directed</td>
</tr>
<tr>
<td><strong>Mesoscale</strong></td>
<td>Invasive tract tracing</td>
<td>Did not discuss</td>
</tr>
<tr>
<td></td>
<td>Directed, weighted</td>
<td></td>
</tr>
<tr>
<td><strong>Macroscale</strong></td>
<td>Diffusion MRI, tractography</td>
<td>fMRI, correlations</td>
</tr>
<tr>
<td></td>
<td>Undirected, weighted</td>
<td>Weighted, can be negative, can be directed</td>
</tr>
</tbody>
</table>

![Brain Graph Diagram](d)
Thresholding and Binarisation

Human functional connectivity matrix from fMRI data.
Every element has a nonzero value.
Thresholding and Binarisation

Matrix after thresholding to retain only the 20% strongest weights.
Thresholding and Binarisation

Matrix after thresholding and binarisation.
Representing an Undirected Graph

- Graph $G = (V, E)$ has two input parameters: $|V| = n, |E| = m$.
  - We define the size of $G$ to be $m + n$. 

We can modify these ideas for directed graphs.
Representing an Undirected Graph

- Graph $G = (V, E)$ has two input parameters: $|V| = n, |E| = m$.
  - We define the size of $G$ to be $m + n$.
- Assume $V = \{1, 2, \ldots, n - 1, n\}$.
- **Adjacency matrix** representation: $n \times n$ Boolean matrix, where the entry in row $i$ and column $j$ is 1 iff the graph contains the edge $(i, j)$.
  - Space used is $O(n^2)$, which is optimal in the worst case.
  - Check if there is an edge between node $i$ and node $j$ in $O(1)$ time.
  - Iterate over all the edges incident on node $i$ in $O(n)$ time.

We can modify these ideas for directed graphs.
Representing an Undirected Graph

- Graph $G = (V, E)$ has two input parameters: $|V| = n, |E| = m$.
  - We define the size of $G$ to be $m + n$.
- Assume $V = \{1, 2, \ldots, n - 1, n\}$.
- **Adjacency matrix** representation: $n \times n$ Boolean matrix, where the entry in row $i$ and column $j$ is 1 iff the graph contains the edge $(i, j)$.
  - Space used is $O(n^2)$, which is optimal in the worst case.
  - Check if there is an edge between node $i$ and node $j$ in
Representing an Undirected Graph

- Graph \( G = (V, E) \) has two input parameters: \(|V| = n, |E| = m\).
  - We define the size of \( G \) to be \( m + n \).
- Assume \( V = \{1, 2, \ldots, n - 1, n\} \).
- **Adjacency matrix** representation: \( n \times n \) Boolean matrix, where the entry in row \( i \) and column \( j \) is 1 iff the graph contains the edge \((i, j)\).
  - Space used is \( O(n^2) \), which is optimal in the worst case.
  - Check if there is an edge between node \( i \) and node \( j \) in \( O(1) \) time.
  - Iterate over all the edges incident on node \( i \) in
Representing an Undirected Graph

- Graph $G = (V, E)$ has two input parameters: $|V| = n, |E| = m$.
  - We define the size of $G$ to be $m + n$.
- Assume $V = \{1, 2, \ldots, n - 1, n\}$.
- *Adjacency matrix* representation: $n \times n$ Boolean matrix, where the entry in row $i$ and column $j$ is 1 iff the graph contains the edge $(i, j)$.
  - Space used is $O(n^2)$, which is optimal in the worst case.
  - Check if there is an edge between node $i$ and node $j$ in $O(1)$ time.
  - Iterate over all the edges incident on node $i$ in $O(n)$ time.

We can modify these ideas for directed graphs.
Representing an Undirected Graph

- Graph $G = (V, E)$ has two input parameters: $|V| = n, |E| = m$.
  - We define the size of $G$ to be $m + n$.
- Assume $V = \{1, 2, \ldots, n - 1, n\}$.
- **Adjacency matrix** representation: $n \times n$ Boolean matrix, where the entry in row $i$ and column $j$ is 1 iff the graph contains the edge $(i, j)$.
  - Space used is $O(n^2)$, which is optimal in the worst case.
  - Check if there is an edge between node $i$ and node $j$ in $O(1)$ time.
  - Iterate over all the edges incident on node $i$ in $O(n)$ time.
- **Adjacency list** representation: array Adj, where Adj[v] stores the list of all nodes adjacent to $v$.
  - An edge $e = (u, v)$ appears twice: in Adj[u] and Adj[v].
Representing an Undirected Graph

- Graph \( G = (V, E) \) has two input parameters: \( |V| = n, |E| = m \).
  - We define the size of \( G \) to be \( m + n \).
- Assume \( V = \{1, 2, \ldots, n - 1, n\} \).
- **Adjacency matrix** representation: \( n \times n \) Boolean matrix, where the entry in row \( i \) and column \( j \) is 1 iff the graph contains the edge \((i, j)\).
  - Space used is \( O(n^2) \), which is optimal in the worst case.
  - Check if there is an edge between node \( i \) and node \( j \) in \( O(1) \) time.
  - Iterate over all the edges incident on node \( i \) in \( O(n) \) time.
- **Adjacency list** representation: array \( \text{Adj} \), where \( \text{Adj}[v] \) stores the list of all nodes adjacent to \( v \).
  - An edge \( e = (u, v) \) appears twice: in \( \text{Adj}[u] \) and \( \text{Adj}[v] \).
  - \( d(v) = \) the number of neighbours of node \( v \).
  - Space used is
Representing an Undirected Graph

- Graph $G = (V, E)$ has two input parameters: $|V| = n$, $|E| = m$.
  - We define the size of $G$ to be $m + n$.
- Assume $V = \{1, 2, \ldots, n - 1, n\}$.
- **Adjacency matrix** representation: $n \times n$ Boolean matrix, where the entry in row $i$ and column $j$ is 1 iff the graph contains the edge $(i, j)$.
  - Space used is $O(n^2)$, which is optimal in the worst case.
  - Check if there is an edge between node $i$ and node $j$ in $O(1)$ time.
  - Iterate over all the edges incident on node $i$ in $O(n)$ time.
- **Adjacency list** representation: array Adj, where Adj[$v$] stores the list of all nodes adjacent to $v$.
  - An edge $e = (u, v)$ appears twice: in Adj[$u$] and Adj[$v$].
  - $d(v) = \text{the number of neighbours of node } v$.
  - Space used is $O(n + \sum_{v \in G} d(v)) = \text{space used}$.
Representing an Undirected Graph

- Graph $G = (V, E)$ has two input parameters: $|V| = n, |E| = m$.
  - We define the size of $G$ to be $m + n$.
- Assume $V = \{1, 2, \ldots, n - 1, n\}$.
- Adjacency matrix representation: $n \times n$ Boolean matrix, where the entry in row $i$ and column $j$ is 1 iff the graph contains the edge $(i, j)$.
  - Space used is $O(n^2)$, which is optimal in the worst case.
  - Check if there is an edge between node $i$ and node $j$ in $O(1)$ time.
  - Iterate over all the edges incident on node $i$ in $O(n)$ time.
- Adjacency list representation: array $\text{Adj}$, where $\text{Adj}[v]$ stores the list of all nodes adjacent to $v$.
  - An edge $e = (u, v)$ appears twice: in $\text{Adj}[u]$ and $\text{Adj}[v]$.
  - $d(v) =$ the number of neighbours of node $v$.
  - Space used is $O(n + \sum_{v \in G} d(v)) = O(n + m)$, which is optimal for every graph.
  - Check if there is an edge between node $u$ and node $v$ in
Representing an Undirected Graph

- Graph $G = (V, E)$ has two input parameters: $|V| = n, |E| = m$.
  - We define the size of $G$ to be $m + n$.
- Assume $V = \{1, 2, \ldots, n - 1, n\}$.
- **Adjacency matrix** representation: $n \times n$ Boolean matrix, where the entry in row $i$ and column $j$ is 1 iff the graph contains the edge $(i, j)$.
  - Space used is $O(n^2)$, which is optimal in the worst case.
  - Check if there is an edge between node $i$ and node $j$ in $O(1)$ time.
  - Iterate over all the edges incident on node $i$ in $O(n)$ time.
- **Adjacency list** representation: array $\text{Adj}$, where $\text{Adj}[v]$ stores the list of all nodes adjacent to $v$.
  - An edge $e = (u, v)$ appears twice: in $\text{Adj}[u]$ and $\text{Adj}[v]$.
  - $d(v) = \text{the number of neighbours of node } v$.
  - Space used is $O(n + \sum_{v \in G} d(v)) = O(n + m)$, which is optimal for every graph.
  - Check if there is an edge between node $u$ and node $v$ in $O(d(u))$ time.
  - Iterate over all the edges incident on node $u$ in
Representing an Undirected Graph

- **Graph** $G = (V, E)$ has two input parameters: $|V| = n, |E| = m$.
  - We define the *size* of $G$ to be $m + n$.
- Assume $V = \{1, 2, \ldots, n - 1, n\}$.
- **Adjacency matrix** representation: $n \times n$ Boolean matrix, where the entry in row $i$ and column $j$ is 1 iff the graph contains the edge $(i, j)$.
  - Space used is $O(n^2)$, which is optimal in the worst case.
  - Check if there is an edge between node $i$ and node $j$ in $O(1)$ time.
  - Iterate over all the edges incident on node $i$ in $O(n)$ time.
- **Adjacency list** representation: array $\text{Adj}$, where $\text{Adj}[v]$ stores the list of all nodes adjacent to $v$.
  - An edge $e = (u, v)$ appears twice: in $\text{Adj}[u]$ and $\text{Adj}[v]$.
  - $d(v) =$ the number of neighbours of node $v$.
  - Space used is $O(n + \sum_{v \in G} d(v)) = O(n + m)$, which is optimal for every graph.
  - Check if there is an edge between node $u$ and node $v$ in $O(d(u))$ time.
  - Iterate over all the edges incident on node $u$ in $O(d(u))$ time.
Representing an Undirected Graph

- Graph $G = (V, E)$ has two input parameters: $|V| = n, |E| = m$.
  - We define the *size* of $G$ to be $m + n$.
- Assume $V = \{1, 2, \ldots, n - 1, n\}$.
- *Adjacency matrix* representation: $n \times n$ Boolean matrix, where the entry in row $i$ and column $j$ is 1 iff the graph contains the edge $(i, j)$.
  - Space used is $O(n^2)$, which is optimal in the worst case.
  - Check if there is an edge between node $i$ and node $j$ in $O(1)$ time.
  - Iterate over all the edges incident on node $i$ in $O(n)$ time.
- *Adjacency list* representation: array Adj, where Adj[$v$] stores the list of all nodes adjacent to $v$.
  - An edge $e = (u, v)$ appears twice: in Adj[$u$] and Adj[$v$].
  - $d(v) = \text{the number of neighbours of node } v$.
  - Space used is $O(n + \sum_{v \in G} d(v)) = O(n + m)$, which is optimal for every graph.
  - Check if there is an edge between node $u$ and node $v$ in $O(d(u))$ time.
  - Iterate over all the edges incident on node $u$ in $O(d(u))$ time.

- We can modify these ideas for directed graphs.
Implementing Hierholzer’s Algorithm

\[ u \leftarrow s \neq u \text{ is the current node.} \]

\[ \textbf{while } d(u) > 0 \text{ do} \]

\[ \text{Output } u. \]

\[ \text{Let } v \text{ be a neighbour of } u. \]

\[ \text{Delete the edge } (u, v) \text{ from } G. \]

\[ u \leftarrow v \]

\[ \textbf{end while} \]
Implementing Hierholzer’s Algorithm

\[ u \leftarrow s \neq u \text{ is the current node.} \]

\[ \textbf{while } d(u) > 0 \text{ do} \]
\[ \quad \text{Output } u. \]
\[ \quad \text{Let } v \text{ be a neighbour of } u. \]
\[ \quad \text{Delete the edge } (u, v) \text{ from } G. \]
\[ \quad u \leftarrow v \]
\[ \textbf{end while} \]
Implementing Hierholzer’s Algorithm

\[ u \leftarrow s \neq u \text{ is the current node.} \]

**while** \( d(u) > 0 \) **do**

Output \( u \).

Let \( v \) be a neighbour of \( u \).

Delete the edge \((u, v)\) from \( G \).

\[ u \leftarrow v \]

**end while**

<table>
<thead>
<tr>
<th>Adjacency matrix</th>
<th>Adjacency list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine ( d(u) )</td>
<td></td>
</tr>
<tr>
<td>Find a neighbour ( v ) of ( u )</td>
<td></td>
</tr>
<tr>
<td>Delete edge ((u, v))</td>
<td></td>
</tr>
</tbody>
</table>
Implementing Hierholzer’s Algorithm

\[ u \leftarrow s \neq u \] is the current node.

\begin{align*}
\textbf{while} & \quad d(u) > 0 \quad \textbf{do} \\
& \quad \text{Output } u. \\
& \quad \text{Let } v \text{ be a neighbour of } u. \\
& \quad \text{Delete the edge } (u, v) \text{ from } G. \\
& \quad u \leftarrow v \\
\textbf{end while}
\end{align*}

<table>
<thead>
<tr>
<th>Adjacency matrix</th>
<th>Adjacency list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine (d(u))</td>
<td>Maintain array of node degrees. (O(1)) time</td>
</tr>
<tr>
<td>Find a neighbour (v) of (u)</td>
<td>Same idea</td>
</tr>
<tr>
<td>Delete edge ((u, v))</td>
<td></td>
</tr>
</tbody>
</table>
Implementing Hierholzer’s Algorithm

\[ u \leftarrow s \ # u \text{ is the current node.} \]

\begin{algorithm*}
\textbf{while} \( d(u) > 0 \) \textbf{do}
\begin{align*}
&\text{Output } u. \\
&\text{Let } v \text{ be a neighbour of } u. \\
&\text{Delete the edge } (u, v) \text{ from } G. \\
&u \leftarrow v
\end{align*}
\textbf{end while}
\end{algorithm*}

<table>
<thead>
<tr>
<th></th>
<th>Adjacency matrix</th>
<th>Adjacency list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine ( d(u) )</td>
<td>Maintain array of node degrees. ( O(1) \text{ time} )</td>
<td>Same idea</td>
</tr>
<tr>
<td>Find a neighbour ( v ) of ( u )</td>
<td>Traverse row for ( u ). ( O(n) \text{ time} )</td>
<td>( v ) is first node in ( \text{Adj}[u] ).</td>
</tr>
<tr>
<td>Delete edge ((u, v))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementing Hierholzer’s Algorithm

Let $u \leftarrow s \neq u$ be the current node.

\begin{align*}
\textbf{while } & d(u) > 0 \textbf{ do} \\
& \text{Output } u. \\
& \text{Let } v \text{ be a neighbour of } u. \\
& \text{Delete the edge } (u, v) \text{ from } G. \\
& u \leftarrow v \\
\end{align*}

Adjacency matrix

<table>
<thead>
<tr>
<th>Determine $d(u)$</th>
<th>Maintain array of node degrees. $O(1)$ time</th>
<th>Same idea</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Find a neighbour $v$ of $u$</th>
<th>Traverse row for $u$. $O(n)$ time.</th>
<th>$v$ is first node in $\text{Adj}[u]$.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Delete edge $(u, v)$</th>
<th>Set both entries to 0; Update $d(v)$. $O(1)$ time.</th>
<th>Delete first element of $\text{Adj}[u]$. Update $d(v)$. $O(1)$ time.</th>
</tr>
</thead>
</table>
Implementing Hierholzer’s Algorithm

\[ u \leftarrow s \neq u \text{ is the current node.} \]

**while** \( d(u) > 0 \) **do**

- Output \( u \).
- Let \( v \) be a neighbour of \( u \).
- Delete the edge \((u, v)\) from \( G \).
- \( u \leftarrow v \)

**end while**

---

<table>
<thead>
<tr>
<th></th>
<th><strong>Adjacency matrix</strong></th>
<th><strong>Adjacency list</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine ( d(u) )</td>
<td>Maintain array of node degrees. ( O(1) ) time</td>
<td>Same idea</td>
</tr>
<tr>
<td>Find a neighbour ( v ) of ( u )</td>
<td>Traverse row for ( u ). ( O(n) ) time.</td>
<td>( v ) is first node in ( \text{Adj}[u] ).</td>
</tr>
<tr>
<td>Delete edge ((u, v))</td>
<td>Set both entries to 0; Update ( d(v) ). ( O(1) ) time.</td>
<td>Delete first element of ( \text{Adj}[u] ). Update ( d(v) ). ( O(1) ) time. <strong>How do we delete ( u ) from ( \text{Adj}[v] )?</strong></td>
</tr>
</tbody>
</table>
Implementing Hierholzer’s Algorithm

\[ u \leftarrow s \neq u \text{ is the current node.} \]

\[
\text{while } d(u) > 0 \text{ do}
\]
\[
\text{Output } u.
\]
\[
\text{Let } v \text{ be a neighbour of } u.
\]
\[
\text{Delete the edge } (u, v) \text{ from } G.
\]
\[
u \leftarrow v
\]
\[
\text{end while}
\]

<table>
<thead>
<tr>
<th>Adjacency matrix (O(n^2)) time.</th>
<th>Adjacency list (O(n)) time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine (d(u))</td>
<td>Maintain array of node degrees. (O(1)) time</td>
</tr>
<tr>
<td>Find a neighbour (v) of (u)</td>
<td>Traverse row for (u). (O(n)) time.</td>
</tr>
<tr>
<td>Delete edge ((u, v))</td>
<td>Set both entries to 0; Update (d(v)). (O(1)) time.</td>
</tr>
</tbody>
</table>

T. M. Murali February 6, 2018 CS 4984: Computing the Brain
Visualising Matrices
Visualising Matrices
Visualising Matrices
Anatomical Projection
Circular Layout
Force-Directed Layout
Node Degree

- Undirected graph \( G = (V, E) \): degree \( d(v) \) of a node \( v \) is the number of edges in \( E \) that are incident on \( v \).

Textbook also defines strength of a node: total weight of edges incident on that node.
Node Degree

- Undirected graph $G = (V, E)$: degree $d(v)$ of a node $v$ is the number of edges in $E$ that are incident on $v$.
  \[ d(v) = |\{u \text{ such that } (u, v) \in E\}| \]

- Directed graph $G = (V, E)$:

(a)
Node Degree

- Undirected graph $G = (V, E)$: degree $d(v)$ of a node $v$ is the number of edges in $E$ that are incident on $v$.
  
  \[ d(v) = |\{u \text{ such that } (u, v) \in E\}| \]

- Directed graph $G = (V, E)$:
  - in-degree $d_{in}(v)$ of node $v$ is the number of edges with $v$ as the head.
  - out-degree $d_{out}(v)$ of node $v$ is the number of edges with $v$ as the tail.

  \[ d_{in}(v) = |\{u \text{ such that } (u, v) \in E\}| \]
  \[ d_{out}(v) = |\{u \text{ such that } (v, u) \in E\}| \]

- Textbook also defines strength of a node: total weight of edges incident on that node.
Node Degree

- Undirected graph $G = (V, E)$: 
  degree $d(v)$ of a node $v$ is the number of edges in $E$ that are incident on $v$. 
  \[ d(v) = |\{u \text{ such that } (u, v) \in E\}| \]

- Directed graph $G = (V, E)$:
  - in-degree $d_{in}(v)$ of node $v$ is the number of edges with $v$ as the head.
  - out-degree $d_{out}(v)$ of node $v$ is the number of edges with $v$ as the tail.
  \[ d_{in}(v) = |\{u \text{ such that } (u, v) \in E\}| \]
  \[ d_{out}(v) = |\{u \text{ such that } (v, u) \in E\}| \]

- Textbook also defines strength of a node: total weight of edges incident on that node.
Degree Distribution

- A way to summarize information about a graph.
**Degree Distribution**

- A way to summarize information about a graph.
- *Degree distribution* of an undirected graph $G$: for every integer $k \geq 0$, the fraction $p(k)$ of nodes in $G$ whose degree is $k$. 

\[ \sum_{k=0}^{\infty} kp(k) = 1 \]

\[ \sum_{k=0}^{\infty} kn(k) = 2m \]

T. M. Murali February 6, 2018 CS 4984: Computing the Brain
Degree Distribution

- A way to summarize information about a graph.
- *Degree distribution* of an undirected graph $G$: for every integer $k \geq 0$, the fraction $p(k)$ of nodes in $G$ whose degree is $k$.
- *Cumulative degree distribution* of $G$: for every integer $k \geq 0$, the fraction $P(k)$ of nodes in $G$ whose degree is at most $k$.
Degree Distribution

- A way to summarize information about a graph.
- *Degree distribution* of an undirected graph $G$: for every integer $k \geq 0$, the fraction $p(k)$ of nodes in $G$ whose degree is $k$.
- *Cumulative degree distribution* of $G$: for every integer $k \geq 0$, the fraction $P(k)$ of nodes in $G$ whose degree is at most $k$.
- Plotting the cumulative degree distribution can offer interesting insights into a graph.

What is the value of $\sum_{k \geq 0} k p(k)$?

Define

$$n(k) = np(k),$$

such that

$$\sum_{k \geq 0} k p(k) = \frac{1}{n} \sum_{k \geq 0} kn(k) = \frac{2}{n} \sum_{v \in V} d(v) = 2m.$$
Degree Distribution

- A way to summarize information about a graph.
- *Degree distribution* of an undirected graph $G$: for every integer $k \geq 0$, the fraction $p(k)$ of nodes in $G$ whose degree is $k$.
- *Cumulative degree distribution* of $G$: for every integer $k \geq 0$, the fraction $P(k)$ of nodes in $G$ whose degree is at most $k$.
- Plotting the cumulative degree distribution can offer interesting insights into a graph.
- What is the value of $\sum_k kp(k)$?
Degree Distribution

- A way to summarize information about a graph.
- *Degree distribution* of an undirected graph $G$: for every integer $k \geq 0$, the fraction $p(k)$ of nodes in $G$ whose degree is $k$.
- *Cumulative degree distribution* of $G$: for every integer $k \geq 0$, the fraction $P(k)$ of nodes in $G$ whose degree is at most $k$.
- Plotting the cumulative degree distribution can offer interesting insights into a graph.
- What is the value of $\sum_k k p(k)$?
- Define $n(k) = np(k)$, the number of nodes with degree $k$.

$$\sum_{k\geq0} k p(k) = \frac{1}{n} \sum_{k\geq0} k n(k)$$
Degree Distribution

- A way to summarize information about a graph.
- **Degree distribution** of an undirected graph $G$: for every integer $k \geq 0$, the fraction $p(k)$ of nodes in $G$ whose degree is $k$.
- **Cumulative degree distribution** of $G$: for every integer $k \geq 0$, the fraction $P(k)$ of nodes in $G$ whose degree is at most $k$.
- Plotting the cumulative degree distribution can offer interesting insights into a graph.
- What is the value of $\sum_k kp(k)$?
- Define $n(k) = np(k)$, the number of nodes with degree $k$.

$$\sum_{k \geq 0} kp(k) = \frac{1}{n} \sum_{k \geq 0} kn(k) = \frac{1}{n} \sum_{v \in V} d(v) = \frac{2m}{n}$$