Graph Representation

- **Graph**
  - An ordered pair $G(V,E)$ with a set of vertices $V$ and a set of edges $E$

- **Extended Graph Representation**
  - Directed vs. undirected graph
    - Whether each edge has a direction
  - Weighted vs. unweighted graph
    - Whether each edge has a weight
  - Labeled vs. unlabeled graph
    - Whether each vertex has a label
  - 2-D vs. 3-D graph representation
    - Each vertex has angles between two linked edges
Why Graph Mining is Important?

- Data are often represented as a graph
  - Biological networks
  - Chemical compounds
  - Internet
  - WWW
  - Electric circuits
  - Workflows
  - Social networks

- Graph is a general model for data mining !!

Graph Data Mining Topics (1)

- Single Graph Mining
  - Frequent sub-graph pattern mining
    - Finding sub-graphs that frequently occur in a graph
  - Graph clustering (Vertex clustering)
    - Partitioning a graph into sub-graphs
  - Vertex classification
    - Classifying a vertex in a graph
Graph Data Mining Topics (2)

- **Graph Dataset Mining**
  - Frequent sub-graph pattern mining
    - Finding sub-graphs that frequently occur among graphs
  - Graph data clustering
    - Grouping similar graphs
  - Graph data classification
    - Classifying a new graph

<table>
<thead>
<tr>
<th>id</th>
<th>graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1.png" alt="Graph 1" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image2.png" alt="Graph 2" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image3.png" alt="Graph 3" /></td>
</tr>
<tr>
<td>4</td>
<td><img src="image4.png" alt="Graph 4" /></td>
</tr>
</tbody>
</table>

Applications

- **Application of Single Graph Mining**
  - Biological network analysis
  - Social network analysis
  - Web community analysis

- **Application of Graph Dataset Mining**
  - Biochemical structure analysis
  - Program control flow analysis
  - XML structure analysis

- **Challenges**
  - Finding the complete set satisfying the minimum support threshold
  - Developing efficient and scalable algorithms
  - Incorporating various kinds of user-specific constraints
Chapters 11 and 13, Graph Data Mining

- **General Definitions**
  - **Graph Clustering**
  - **Subgraph Pattern Mining**

---

**Connectivity**

- **Degree, \( \text{deg}(v_i) \)**
  - The number of links from \( v_i \) to other vertices
  - Incoming degree and outgoing degree for directed graphs
  - Weighted degree (sum of the weights of the edges directly connected) for weighted graphs

- **A Set of Neighbors, \( N(v_i) \)**
  - A set of vertices directly linked to the vertex \( v_i \)
  - Also called adjacent neighbors or direct neighbors

- **Degree Distribution, \( P(k) \)**
  - Probability that a vertex has exactly \( k \) links
  - The number of vertices whose degree is \( k \) over the total number of vertices
Length & Size

- **Walk**
  - A sequence of vertices such that each is linked to its succeeding one

- **Path**
  - A walk such that each vertex in the walk is distinct

- **Path Length**
  - The number of edges in path \( p \)

- **Shortest Path between \( v_i \) and \( v_j \)**
  - A path with the smallest length out of all paths from \( v_i \) to \( v_j \)

- **Characteristic Path Length of \( G \)**
  - Average length of the shortest paths between each pair of vertices

- **Diameter of \( G \)**
  - Largest length of the shortest paths between each pair of vertices

Density

- **Density of \( G(V,E) \)**
  - The number of actual edges in \( G \) over the number of all possible edges
  - \( D(G) = \frac{2|E|}{|V||(V|-1)} \)

- **Clique**
  - A fully connected graph (also called, complete graph)
  - \( D(G) = 1 \)

- **Quasi-Clique**
  - Close to clique
  - A densely connected sub-graph
  - \( D(G) > \theta \) where \( \theta \) is a user-specified threshold
Modularity

- **Clustering Coefficient of** $v_i$
  - The density of a sub-graph $G'(V', E')$ where $V'$ is the set of neighbors of $v_i$
  - $C(v_i) = \frac{\sum_{j \in N(v_i)} \binom{|N(v_j)|}{2}}{|N(v_i)|(|N(v_i)|-1)}$
  - Measuring the effectiveness of $v_i$ on denseness

- **Average Clustering Coefficient of** $G(V,E)$
  - Average of the clustering coefficients of all vertices in $V$
  - Maximum is 1
  - Measuring the modularity of $G$

Centrality

- **Closeness, $C_c(v_i)$**
  - Detects the vertices located in the center of a graph
  - $C_c(v_i) = \frac{1}{\sum_{j \neq i} |p_i(v_i, v_j)|}$
  - where $|p_i(v_i, v_j)|$ is the shortest path length between $v_i$ and $v_j$

- **Betweenness, $C_b(v_i)$**
  - Detects the vertices located between two clusters
  - $C_b(v_i) = \sum_{s \neq i \neq t} \frac{\sigma_s(v_i)}{\sigma_{st}}$
  - where $\sigma_{st}$ is the number of shortest paths between $s$ and $t$, and $\sigma_s(v_i)$ is the number of shortest paths between $s$ and $t$, which pass through the vertex $v_i$
Chapters 11 and 13, Graph Data Mining

- General Definitions
  - Graph Clustering
  - Subgraph Pattern Mining

Graph Clustering

- Problem Definition
  - Finding densely connected sub-graphs $G'(V',E')$ from a graph $G(V,E)$
    - Finding sub-graphs with dense intra-connections and sparse interconnections (modularity)

- Methods
  - Density-based methods
  - Hierarchical methods
  - Partition-based methods
Chapters 11 and 13, Graph Data Mining

- General Definitions
- Graph Clustering
  - Density-based Methods
  - Partition-based Methods
  - Hierarchical Methods
- Subgraph Pattern Mining

Maximum Clique

- Algorithm
  - Find all maximum sized cliques
  - Use antimonotonic property
    - If a subset of set A is not a clique, then the set A is not a clique

size 2 cliques: 
{AB}, {AC}, {AE}, ..... 

size 3 cliques: 
{ABC}, {ACE}, ..... 

size 4 cliques: {JKLM}
Clique Percolation

- **Definitions**
  - Two $k$-cliques are adjacent if they share $(k-1)$ vertices where $k$ is the number of vertices in each clique.
  - A $k$-clique chain is a sub-graph comprising the union of a sequence of adjacent $k$ cliques.

- **Algorithm**
  1. Find all $k$-cliques.
  2. Find all maximal $k$-clique chains by iterative merging adjacent $k$-cliques.

- **Reference**

---

$k$-Core Decomposition

- **Definition**
  - $k$-core is a sub-graph by pruning all vertices whose degree is less than $k$.

- **Algorithm**
  - Remove repeatedly all vertices whose degree is less than $k$.

- **Reference**
Seed Growth

- **Main Idea**
  - Search for local optimization
  - Use a modularity (or density) function
  - Types of seeds
    - Random seeds: selected randomly
    - Core seeds: selected by degree or clustering coefficient

- **Algorithm**
  1. Select a vertex (seed) as an initial cluster $S$
  2. Add a neighbor of the seed to $S$ repeatedly to find the maximum modularity (or density)
  3. Return $S$ if the modularity of $S >$ threshold
  4. Repeat (1), (2) and (3) to find a set of clusters

Graph Entropy (1)

- **Definitions**
  - An example of seed-growth algorithms
  - General notation
    - Inner links of $v$ in $G'(V',E')$: edges from $v$ to the vertices in $V'$
      \[ p_i(v) \] probability of $v$ having inner links
    - Outer links of $v$ in $G'(V',E')$: edges from $v$ to the vertices not in $V'$
      \[ p_o(v) \] probability of $v$ having outer links
  - Definitions
    - Vertex entropy: $e(v) = -p_i(v) \log_2 p_i(v) - p_o(v) \log_2 p_o(v)$
    - Graph entropy: $e(G(V,E)) = \sum_{v \in V} e(v)$
  - Find the minimum graph entropy during seed growth
Graph Entropy (2)

Example

Algorithm
(1) Select a seed node, and include all neighbors of the seed node into a seed cluster
(2) Iteratively remove a neighbor if removal decreases graph entropy
(3) Iteratively add a node on the outer boundary of a current cluster if addition decreases graph entropy
(4) Output the cluster with the minimal graph entropy
(5) Repeat (1), (2), (3), and (4) until no seed node remains

Reference
Chapters 11 and 13, Graph Data Mining

- General Definitions
- Graph Clustering
  - Density-based Methods
  - Partition-based Methods
  - Hierarchical Methods
- Subgraph Pattern Mining

Restricted Neighborhood Search (1)

- Main Idea
  - Iterative moves of vertices to find the best global modularity
  - Types of moves
    - Global move: moving a random vertex to a random cluster
    - Intensification move: moving in the restricted neighborhood (vertices on the boundary of partitions)

- Algorithm
  1. Randomly partition the graph into $k$ subgraphs
  2. Make an intensification move of a random vertex if this move improves modularity
  3. Repeat (2) until finding the best modularity
Restricted Neighborhood Search (2)

- **Example**
  - Modularity function: number of interconnecting edges between clusters

- **Reference**
  - King, A., et al., "Protein complex prediction via cost-based clustering"

---

CSI 4352, Introduction to Data Mining

**Chapters 11 and 13, Graph Data Mining**

- **General Definitions**
- **Graph Clustering**
  - Density-based Methods
  - Partition-based Methods
  - Hierarchical Methods
- **Subgraph Pattern Mining**
Bottom-Up vs. Top-Down

- **Bottom-Up (Agglomerative) Approaches**
  - Start with each vertex as a cluster
  - Iteratively merge the closest clusters
  - Require a distance function between two clusters

- **Top-Down (Divisive) Approaches**
  - Start with the whole graph as a cluster
  - Recursively divide up the clusters
  - Require a cutting algorithm

Merging by Shortest Path Length

- **Main Idea**
  - Agglomerative approach using single-link distance

- **Algorithm**
  1. Select two closest vertices from different clusters based on the shortest path length between them
  2. Merge two clusters that include the selected vertices
  3. Repeat (1) and (2) until the shortest path length reaches a threshold
Merging by Common Neighbors (1)

- **Main Idea**
  - Agglomerative approach using the similarity based on common neighbors
  - More common neighbors two vertices share, more similar they are

- **Algorithm**
  1. Find the most similar vertices from different clusters based on a similarity function
  2. Merge the two clusters if the merged cluster reaches a density threshold
  3. Repeat (1) and (2) until no more clusters can be merged

- **Similarity Functions**
  - Jaccard coefficient: \( S(x, y) = \frac{|N(x) \cap N(y)|}{|N(x) \cup N(y)|} \)
  - Geometric coefficient: \( S(x, y) = \frac{|N(x) \cap N(y)|^2}{|N(x)| \cdot |N(y)|} \)

Merging by Common Neighbors (2)

- **More Similarity Functions**
  - Dice coefficient: \( S(x, y) = \frac{2|N(x) \cap N(y)|}{|N(x)| + |N(y)|} \)
  - Simpson coefficient: \( S(x, y) = \frac{|N(x) \cap N(y)|}{\min(|N(x)|, |N(y)|)} \)
  - Maryland bridge coefficient: \( S(x, y) = \frac{1}{2} \left( \frac{|N(x) \cap N(y)|}{|N(x)|} + \frac{|N(x) \cap N(y)|}{|N(y)|} \right) \)

- **Reference**
Merging by Statistical Significance

**Statistical Similarity Function**
- Hyper-geometric coefficient \( (P\text{-value}) \):
  \[
  P = \frac{V}{Z} \times \frac{V-Z}{X-Z} \times \frac{V-X}{Y-Z}
  \]
  \( V \) is the total number of vertices,
  \( X = |N(x)| \), \( Y = |N(y)| \), and
  \( Z = |N(x) \cap N(y)| \) for vertices \( x \) and \( y \)

**Algorithm**
- Repeatedly find the vertices with the smallest \( P \)-value, and merge them

**Reference**

---

Minimum Cut

**Definitions**
- Cut: a set of edges whose removal disconnects the graph
- Minimum cut: a cut with minimum number of edges

**Algorithm**
- Recursively find the minimum cut

**Parameter**
- Minimum density threshold
- Minimum size threshold
Betweenness Cut

- **Betweenness**
  - Measurement of vertices or edges located between clusters

- **Algorithm**
  1. Iteratively eliminate a vertex or an edge with the highest Betweenness value until the graph is separated
  2. Recursively apply (1) into each subgraph
  3. Repeat (1) and (2) until all subgraphs reach a density threshold

- **Reference**

Dividing by Common Neighbors

- **Main Idea**
  - Divisive approach using the dissimilarity based on common neighbors
  - Less common neighbors two vertices share, more dissimilar they are

- **Algorithm**
  1. Iteratively eliminate the edge between the most dissimilar vertices based on a similarity function, until the graph is separated
  2. Recursively apply (1) into each subgraph
  3. Repeat (1) and (2) until all subgraphs reach a density threshold

- **Reference**
Chapters 11 and 13, Graph Data Mining

- General Definitions
- Graph Clustering
  - Subgraph Pattern Mining

Properties of Subgraph Patterns from Graph Dataset

- Properties
  - Anti-monotonic property → Apriori algorithm
  - If a sub-graph G is not frequent, then none of the super-graphs of G are frequent

- Example
  - If is infrequent,
  - so do
  - and

<table>
<thead>
<tr>
<th>id</th>
<th>graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1" alt="Graph 1" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image2" alt="Graph 2" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image3" alt="Graph 3" /></td>
</tr>
<tr>
<td>4</td>
<td><img src="image4" alt="Graph 4" /></td>
</tr>
</tbody>
</table>
Properties of Subgraph Patterns from a Single Graph

Properties
- Frequent sub-graph pattern mining in a graph → Not anti-monotonic property!
- Even if a sub-graph G is not frequent, some of the super-graphs of G might be frequent

Example
- Suppose minimum support is 10
- How many ?
- How many ?

FSG Algorithm

FSG
- Frequent Sub-graph discovery

Algorithm
1. Initially, find all frequent size-3 sub-graphs
2. Generate candidate size-(k+1) sub-graphs from frequent size-k sub-graphs
3. Count support of each candidate sub-graph to select frequent sub-graphs
4. Repeat (2) and (3) until no frequent sub-graph or no candidate is found
Candidate Sub-graphs

- **Selective Joining**
  - Join two size-k sub-graphs if they share size-(k-1) sub-graphs
  - May join the same size-k sub-graph
  - Produce multiple distinct size-(k+1) sub-graphs

```
+ + →
```

```
+ + →
```

Summary of FSG Algorithm

- **Strength**
  - Apriori pruning

- **Weakness**
  - Generates a huge set of candidate sub-graphs
  - Requires multiple scans of database
  - Inefficient for mining large-sized sub-graph patterns
  - Needs efficient finding of isomorphic graphs to count support

- **Reference**
Structural Isomorphism

- **Definition**
  - If two graphs are isomorphic, then they are structurally identical

- **Examples**

```
  x   y   z
  |   |   |
  x   |   x
  |   |   |
  z   x   z
```

Canonical Adjacency Matrix

- **Adjacency Matrix**
  - 

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Canonical Adjacency Matrix**

```
  x   y   z
  |   |   |
  x   |   x
  |   |   |
  z   x   z
```

- **Canonical Code**
  - x1110x100y11z0z
FFSM Algorithm

**FFSM**
- Fast frequent sub-graph mining

**Main Idea**
- Use canonical adjacency matrices for selective joining and support counting

**Reference**

---

**DFS Code**

**DFS Tree**
- Trace on depth-first-search

**DFS Tree in a Lexicographic Order**
- DFS tree constructed by the lexicographic order of labels

**DFS Code**
- A sequence of edges of the DFS tree in a lexicographic order
  - \{((x,x), (x,y), (y,x), (y,z), (z,x), (y,z))\}
gSpan Algorithm

- **gSpan**
  - Graph-based sub-structure pattern mining

- **Main Idea**
  - Use DFS codes for frequent sub-graph pattern mining
  - Efficient for pattern growth

- **Reference**
  - Yan, X. and Han, J., “gSpan: Graph-based sub-structure pattern mining”, *In Proceedings of ICDM* (2002)

Questions?

- Lecture Slides on the Course Website,
  "www.ecs.baylor.edu/faculty/cho/4352"