

# Graph Data Mining

**Young-Rae Cho, Ph.D.**

Associate Professor

Division of Software / Division of Digital Healthcare

Yonsei University – Mirae Campus

# 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
- 0-D vs. 1-D vs. 2-D vs. 3-D graph representation
  - Whether each vertex has a specific coordinate



## Why Graph Data Mining is Important?

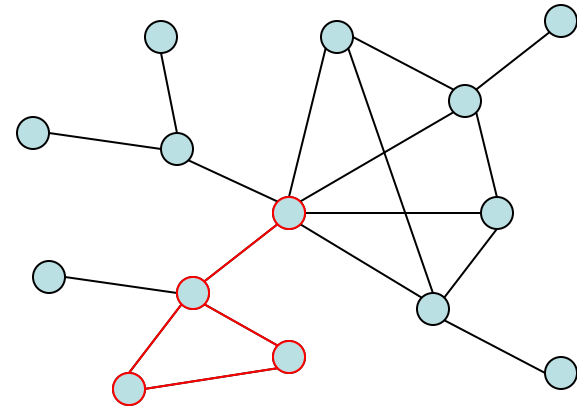
- ❑ **Data are often represented as a graph**
  - Biological networks
  - Chemical compounds
  - Internet
  - WWW
  - Electric circuits
  - Workflows
  - Social networks
  
- ❑ **Graph is a general model for data mining !!**

# 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

id	graph
1	
2	
3	
4	

# Applications



## ❑ Applications of Single Graph Mining

- Biological network analysis
- Social network analysis
- Web community analysis

## ❑ Applications 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

# Overview



1. **General Definitions**
2. **Graph Clustering**
3. **Subgraph Pattern Mining**

# Connectivity



## □ Degree

- Degree of a vertex,  $\deg(v_i)$ : 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

- A set of neighbors of a vertex,  $N(v_i)$ : a set of vertices directly linked to the vertex  $v_i$
- Also called adjacent neighbors or direct neighbors

## □ Degree Distribution

- Degree distribution of a graph  $G$ : Probability that a vertex in  $G$  has exactly  $k$  links,  $P(k)$
- The number of vertices whose degree is  $k$  over the total number of vertices in  $G$



## 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

- Characteristic path length of a graph  $G$ : average length of the shortest paths between each pair of vertices in  $G$

### ❑ Diameter

- Diameter of a graph  $G$ : largest length of the shortest paths between each pair of vertices in  $G$

# Density



## □ Density

- Density of a graph  $G$ : the number of actual edges in  $G$  over the number of all possible edges

- $$D(G) = \frac{2|E|}{|V|(|V|-1)}$$
 where  $G(V,E)$

- The range of density?

## □ 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

## ❑ Clustering Coefficient

- Clustering coefficient of a vertex  $v_i$ : The density of a sub-graph  $G'(V',E')$  where  $V'$  is the set of neighbors of  $v_i$

- $$C(v_i) = \frac{|\bigcup_{v_j, v_k \in N(v_i)} \{\langle v_j, v_k \rangle\}|}{|N(v_i)|(|N(v_i)| - 1)}$$

- The range of a clustering coefficient?
- Measuring the effectiveness of  $v_i$  on denseness

## ❑ Average Clustering Coefficient

- Average clustering coefficient of a graph  $G$ : the average of the clustering coefficients of all vertices in  $G$
- The range of an average clustering coefficient?
- Measuring the modularity of  $G$

## □ Closeness

- Closeness of a vertex  $v_i$ ,  $C_C(v_i)$ : the inverse of the sum of shortest path length between  $v_i$  and all vertices in the graph
- $C_C(v_i) = \frac{1}{\sum_{v_j \in V} |p_s(v_i, v_j)|}$  where  $|p_s(v_i, v_j)|$  is the shortest path length between  $v_i$  and  $v_j$
- Detects the vertices located in the center of a graph

## □ Betweenness

- Betweenness of a vertex  $v_i$ ,  $C_B(v_i)$ : the sum of the ratios of the shortest paths which pass through  $v_i$
- $C_B(v_i) = \sum_{s \neq v_i \neq t \in V} \frac{\sigma_{st}(v_i)}{\sigma_{st}}$  where  $\sigma_{st}$  is the number of shortest paths between  $s$  and  $t$ , and  $\sigma_{st}(v_i)$  is the number of shortest paths between  $s$  and  $t$ , which pass through  $v_i$
- Detects the vertices located between two clusters

# Overview



1. **General Definitions**
2. **Graph Clustering**
3. **Subgraph Pattern Mining**

# Graph Clustering



## □ Problem Definition

- Finding a set of sub-graphs  $G'(V',E')$  from a graph  $G(V,E)$ 
  - Clusters: the sub-graphs
  - Clustering criteria: dense intra-connections and sparse interconnections between the sub-graphs → modularity
- Components (vertices or edges) are overlapping vs. non-overlapping?

## □ Methods

- Density-based methods
- Partition-based methods
- Hierarchical methods

# Overview



1. **General Definitions**
2. **Graph Clustering**
  - **Density-Based Methods**
  - **Partition-Based Methods**
  - **Hierarchical Methods**
3. **Subgraph Pattern Mining**

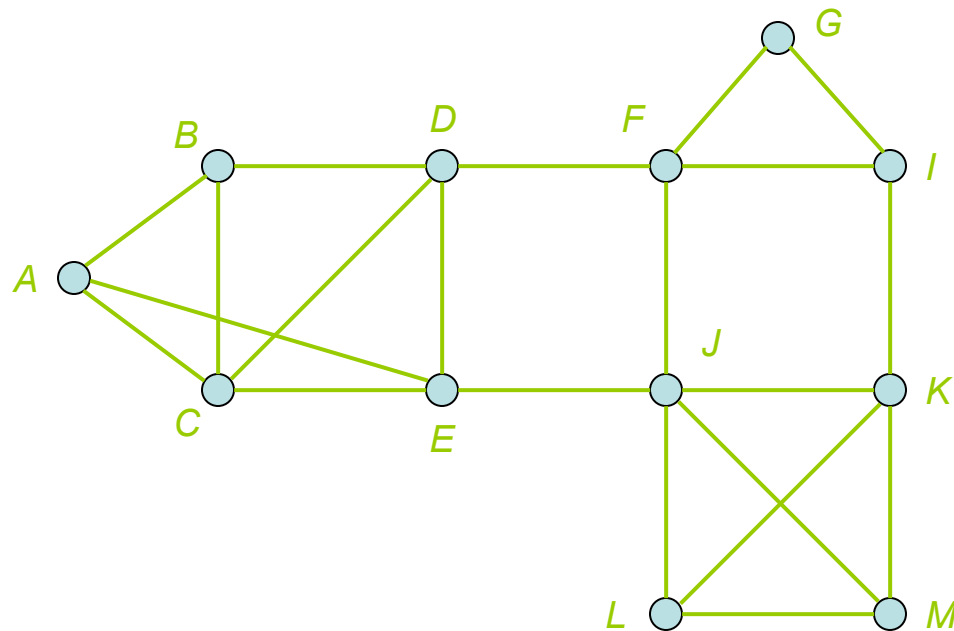
# Maximal Clique Algorithm

## □ Main Idea

- Find all maximum-sized cliques
- How to find all maximum-sized cliques?

## □ Example

- Use of the anti-monotonic property



Size-2 cliques:

$\{AB\}, \{AC\}, \{AE\}, \dots$

Size-3 cliques:

$\{ABC\}, \{ACE\}, \dots$

Size-4 cliques:  $\{JKLM\}$



# Clique Percolation Algorithm



## ❑ 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

## ❑ Process

- 1) Find all  $k$ -cliques
- 2) Find all maximal  $k$ -clique chains by iterative merging adjacent  $k$ -cliques

## ❑ Reference

- Palla, G., et al., “Uncovering the overlapping community structure of complex networks in nature and society”, Nature (2005)

# k-Core Decomposition Algorithm

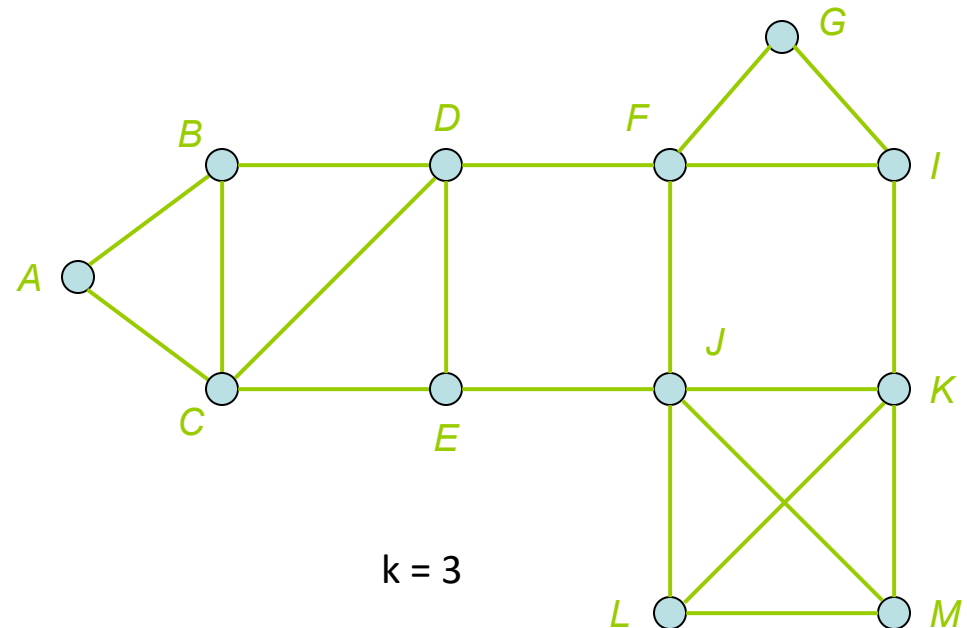


## □ Definition

- k-core is a sub-graph by pruning all vertices whose degree is less than k

## □ Process

- Remove repeatedly all vertices whose degree is less than k to find a set of k-cores



## □ Reference

- Wuchy, S. and Almaas, E., “Peeling the yeast protein network”, Proteomics (2005)

# Seed Growth Algorithms



## □ Main Idea

- Search for local optimization from a seed vertex  
→ Local greedy algorithm
- Grow a sub-graph from a seed vertex to optimize a modularity (density) function
- Types of seed vertices
  - Random seeds: selected randomly
  - Core seeds: selected by degree or clustering coefficient

## □ Process

- 1) Select a vertex (seed) as an initial cluster  $S$
- 2) Add a neighbor of a vertex in  $S$  repeatedly if addition increases modularity
- 3) Return  $S$  if modularity does not increase or modularity  $>$  threshold
- 4) Repeat (1), (2) and (3) to find a set of clusters



# Graph Entropy Algorithm (1)

## □ Main Idea

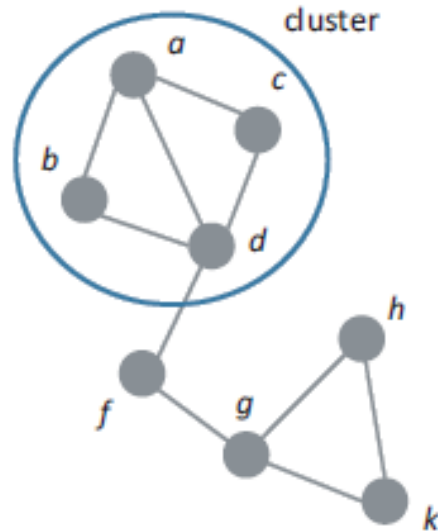
- An example of seed-growth algorithms
- Use Graph Entropy as the modularity function
- Find the minimum graph entropy during seed growth

## □ Definitions

- Inner links, Outer links
  - 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
- Vertex entropy:  $e(v) = - p_i(v) \log_2 p_i(v) - p_o(v) \log_2 p_o(v)$
- Graph entropy :  $e(G(V,E)) = \sum_{v \in V} e(v)$

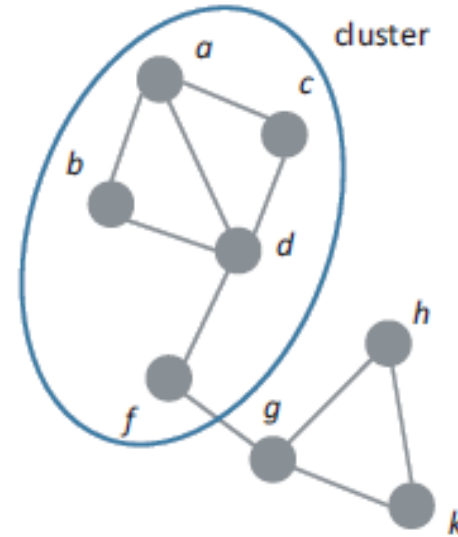
# Graph Entropy Algorithm (2)

## Example



$$\begin{aligned}
 e(a) &= e(b) = e(c) = 0 \\
 e(d) &= 0.81 \\
 e(f) &= 1.00 \\
 e(g) &= e(h) = e(k) = 0
 \end{aligned}$$

(a)



$$\begin{aligned}
 e(a) &= e(b) = e(c) = e(d) = 0 \\
 e(f) &= 1.00 \\
 e(g) &= 0.92 \\
 e(h) &= e(k) = 0
 \end{aligned}$$

(b)

## Graph Entropy Algorithm (3)



### □ Process

- 1) Select a seed vertex, and include all neighbors of the seed vertex into a seed cluster
- 2) Iteratively remove a neighbor if removal decreases graph entropy
- 3) Iteratively add a vertex 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 vertex remains

### □ Reference

- Kenley, E.C. and Cho, Y.-R., “Detecting protein complexes and functional modules from protein interaction networks: a graph entropy approach”, Proteomics (2011)

# Overview



1. **General Definitions**
2. **Graph Clustering**
  - **Density-Based Methods**
  - **Partition-Based Methods**
  - **Hierarchical Methods**
3. **Subgraph Pattern Mining**

# Restricted Neighborhood Search (1)



## □ Main Idea

- Random partition and 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)

## □ Process

- 1) Randomly partition the graph into  $k$  sub-graphs
- 2) Make an intensification move of a random vertex if this move improves modularity
- 3) Repeat (2) until finding the best modularity

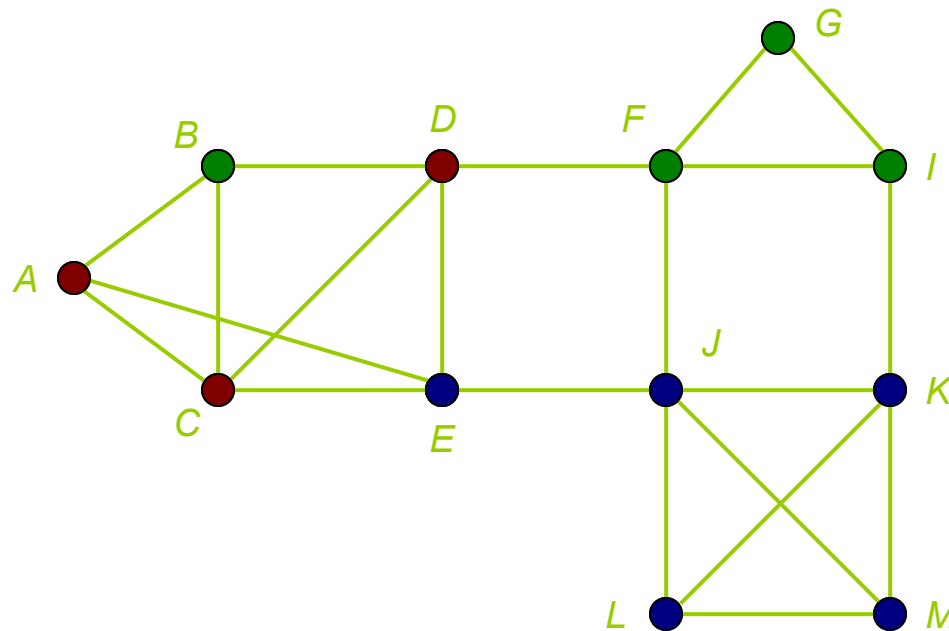


## Restricted Neighborhood Search (2)



### □ Example

- Use the number of interconnecting edges between clusters as a modularity function



### □ Reference

- King, A., et al., "Protein complex prediction via cost-based clustering" Bioinformatics (2004)

# Overview



1. **General Definitions**
2. **Graph Clustering**
  - **Density-Based Methods**
  - **Partition-Based Methods**
  - **Hierarchical Methods**
3. **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

## □ Process

- 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



## □ Main Idea

- Agglomerative approach using similarity based on common neighbors  
→ More common neighbors two vertices share, more similar they are

## □ Process

- 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:

$$S(x, y) = \frac{|N(x) \cap N(y)|^2}{|N(x)| \cdot |N(y)|}$$

- Dice coefficient:  $S(x, y) = \frac{2|N(x) \cap N(y)|}{|N(x)| + |N(y)|}$

- Simpson coefficient:

$$S(x, y) = \frac{|N(x) \cap N(y)|}{\min(|N(x)|, |N(y)|)}$$



## Merging by Statistical Significance

### ❑ Statistical Similarity Function

- Hyper-geometric P-value:

$$P = \frac{\binom{V}{Z} \binom{V-Z}{X-Z} \binom{V-X}{Y-Z}}{\binom{V}{X} \binom{V}{Y}}$$

V is the total number of vertices,

X = |N(x)|,

Y = |N(y)|,

Z = |N(x) ∩ N(y)| for vertices x and y

### ❑ Process

- 1) Find the vertices with the smallest P-value
- 2) Merge two clusters that include the selected vertices
- 3) Repeat (1) and (2) until no more clusters can be merged

### ❑ Reference

- Samanta, M.P. and Liang, S., “Predicting protein functions from redundancies in large-scale protein interaction networks” PNAS (2003)

# Minimum Cut

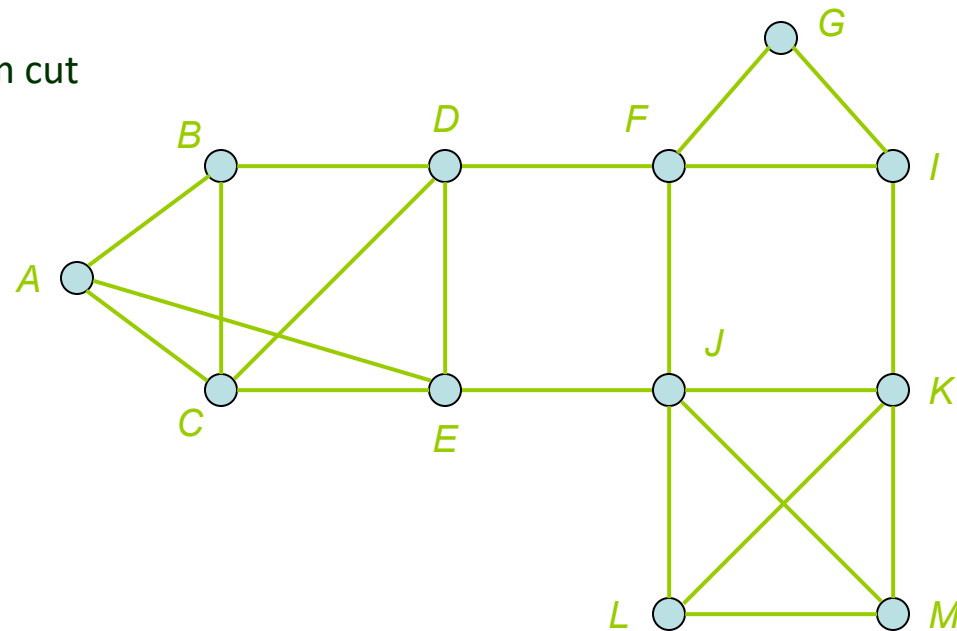


## ❑ Definitions

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

## ❑ Process

- Recursively find the minimum cut



## ❑ Required Parameters

- Minimum density threshold
- Minimum size threshold

# Betweenness Cut



## ❑ Definitions

- Betweenness of a vertex: measurement of vertices located between clusters
- Betweenness of an edge: measurement of edges located between clusters

## ❑ Process

- 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

- Dunn, R., et al., “The use of edge-betweenness clustering to investigate biological function in protein interaction networks” BMC Bioinformatics (2005)



# 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

## ❑ Process

- 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

- Radicchi, F., et al., “Defining and identifying communities in networks” PNAS (2004)

# Overview



1. **General Definitions**
2. **Graph Clustering**
3. **Subgraph Pattern Mining**

# Subgraph Patterns from Graph Dataset

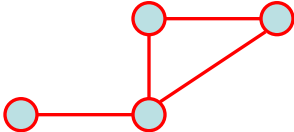


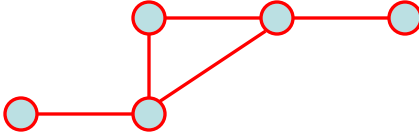
## Properties

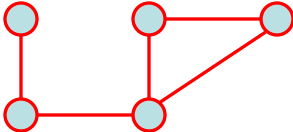
- Anti-monotonic property → Apriori algorithm
- If a sub-graph G is not frequent, then none of the super-graphs of G are frequent

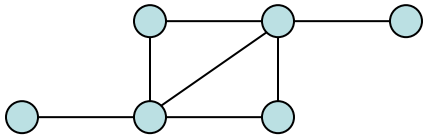
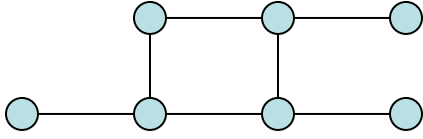
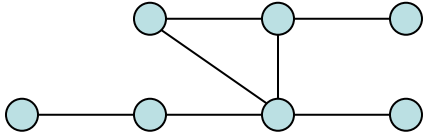
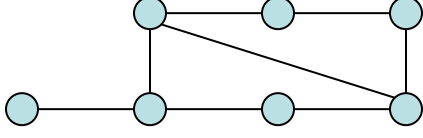
## Example

- Suppose minimum support is 75%

- If  is infrequent,

so do 

and 

id	graph
1	
2	
3	
4	

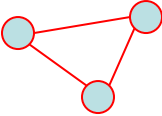
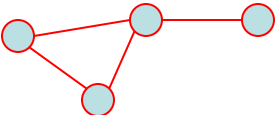
# Subgraph Patterns from a Single Graph

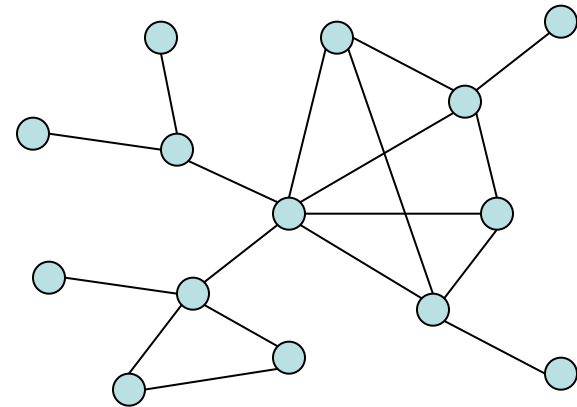
## □ Properties

- Frequent sub-graph pattern mining in a graph  
→ Not follow the 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 (Frequent Sub-Graph discovery)



## □ Main Idea

- Apply the Apriori-like process to graph datasets to detect frequent sub-graph patterns

## □ Process

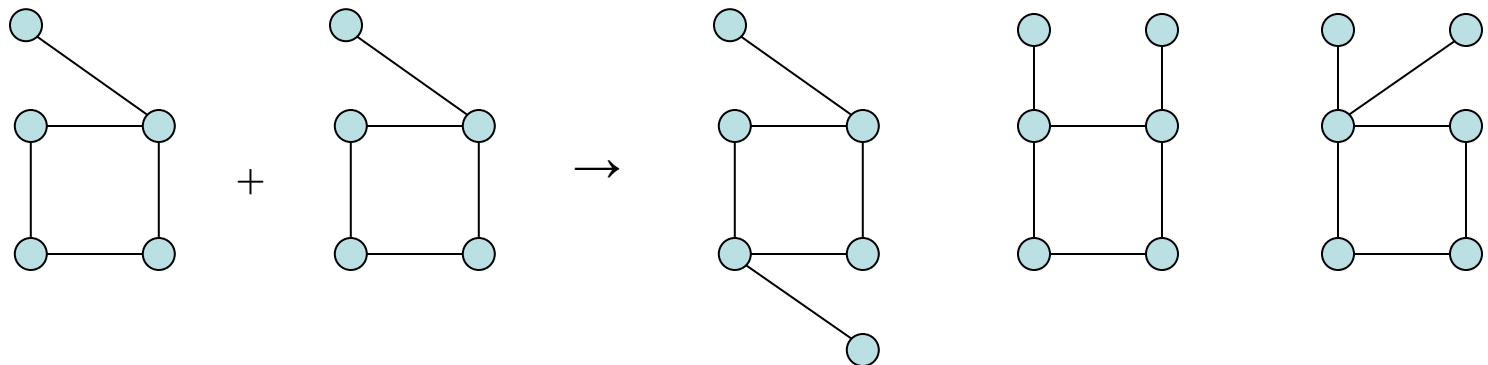
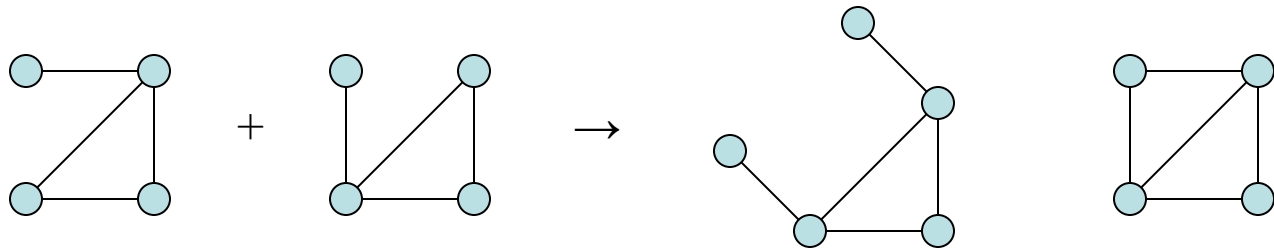
- 1) Initially, find all frequent size-3 sub-graphs
- 2) Generate candidate size-(k+1) sub-graphs from frequent size-k sub-graphs
- 3) Count support of each candidate sub-graph to select frequent sub-graphs
- 4) Repeat (2) and (3) until no frequent sub-graph or no candidate is found

# Generating Candidate Sub-Graphs



## □ Selective Joining

- (Main idea) Join two size- $k$  sub-graphs if they share a size- $(k-1)$  sub-graph
- Join the same size- $k$  sub-graphs too
- Produce multiple distinct size- $(k+1)$  sub-graphs



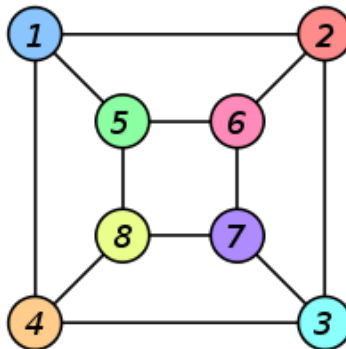
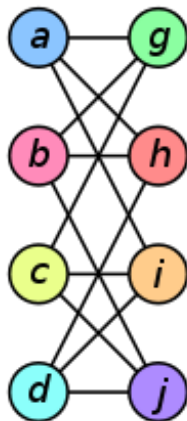
# Counting Support of Sub-Graphs

## Support Computation

- Detect any isomorphic structure of each candidate sub-graph in the graph dataset

## Isomorphic Graphs

- If two graphs are isomorphic, then they are structurally identical
- Example



$$f(a) = 1$$

$$f(b) = 6$$

$$f(c) = 8$$

$$f(d) = 3$$

$$f(g) = 5$$

$$f(h) = 2$$

$$f(i) = 4$$

$$f(j) = 7$$

# 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

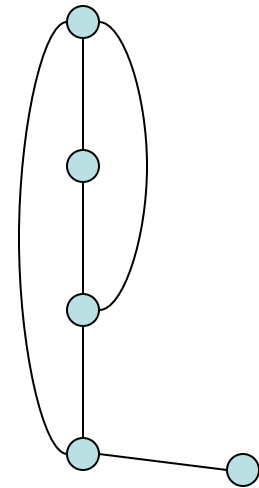
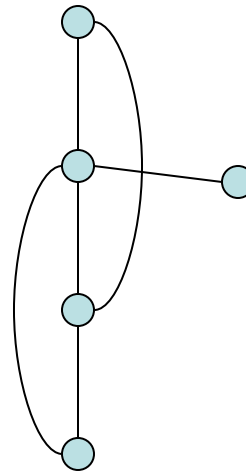
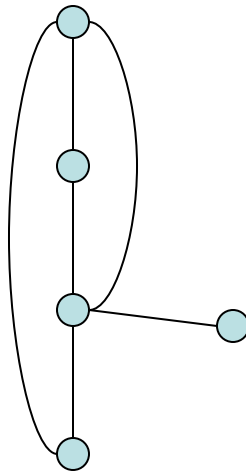
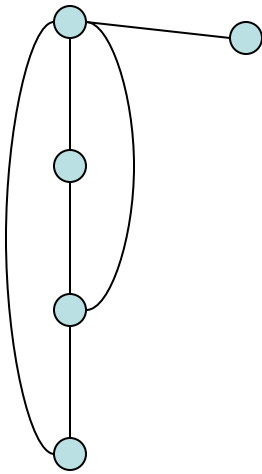
- Kuramochi, M. and Karypis, G., “Frequent subgraph discovery.” In Proceedings of ICDM (2001)



# Structural Isomorphism of Unlabeled Graphs



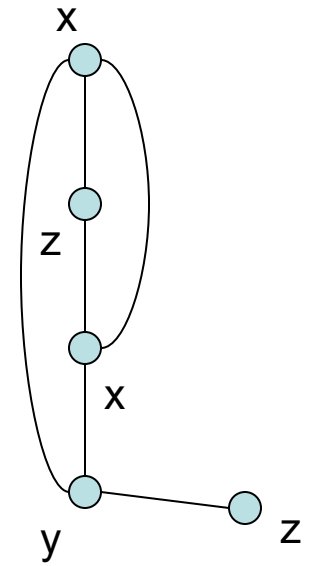
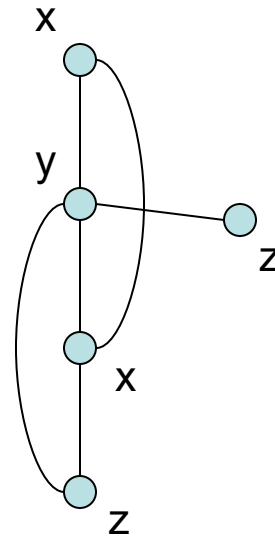
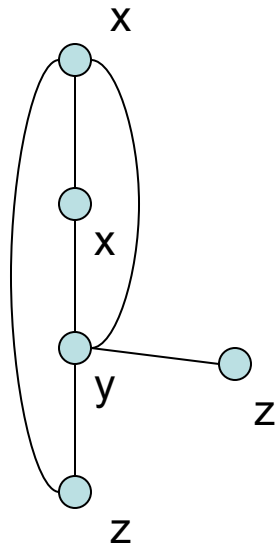
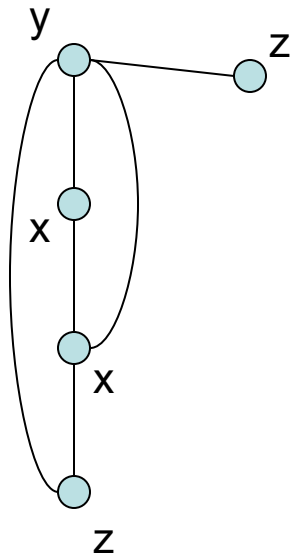
## □ Examples



# Structural Isomorphism of Labeled Graphs



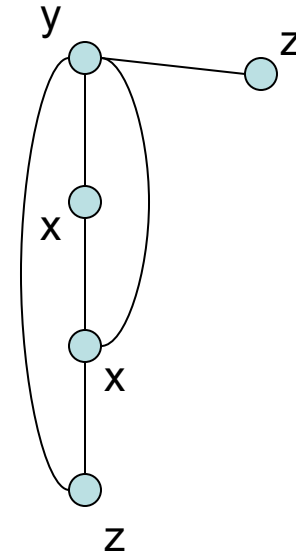
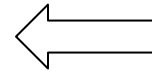
## □ Examples



# Canonical Adjacency Matrix

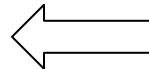
## Canonical Adjacency Matrix

	x	x	y	z	z
x	0	1	1	1	0
x	1	0	1	0	0
y	1	1	0	1	1
z	1	0	1	0	0
z	0	0	1	0	0



## Canonical Code

- x1110x100y11z0z



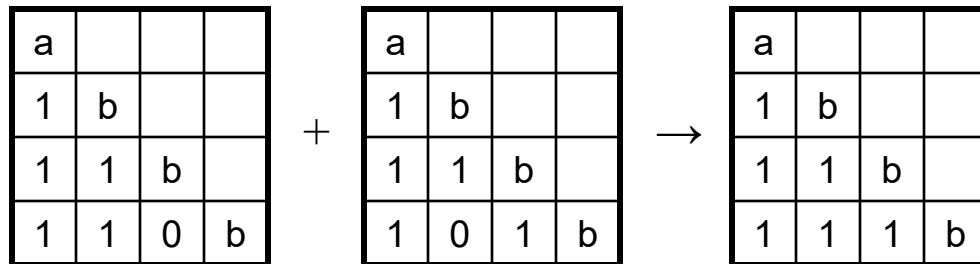
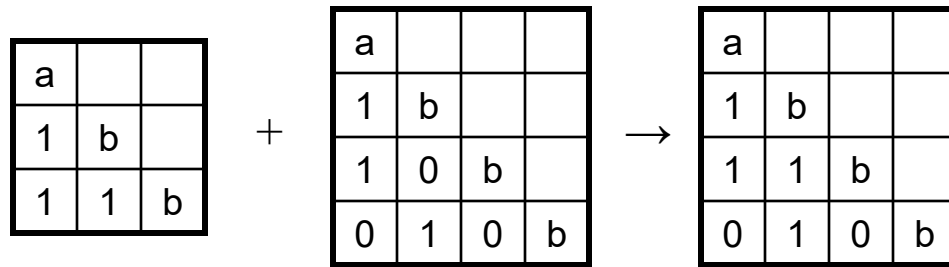
x				
1	x			
1	1	y		
1	0	1	z	
0	0	1	0	z

# FFSM (Fast frequent sub-graph mining)



## □ Main Idea

- Use canonical adjacency matrices for selective joining and support counting



## □ Reference

- Huan, J., Wang, W. and Prins, J., "Efficient mining of frequent subgraph in the presence of isomorphism." In Proceedings of ICDM (2003)

## Questions?



- ❑ Lecture Slides on the Course Website, “[https://ads.yonsei.ac.kr/faculty/data\\_mining](https://ads.yonsei.ac.kr/faculty/data_mining)”

