CSE 590
Data Science Fundamentals

Graph data Mining

Klaus Mueller

Computer Science Department
Stony Brook University and SUNY Korea

Lecture	Topic	Projects
1	Intro, schedule, and logistics	
2	Data Science components and tasks	
3	Data types	Project #1 out
4	Introduction to R, statistics foundations	
5	Introduction to D3, visual analytics	
6	Data preparation and reduction	
7	Data preparation and reduction	Project #1 due
8	Similarity and distances	Project #2 out
9	Similarity and distances	
10	Cluster analysis	
11	Cluster analysis	
12	Pattern miming	Project #2 due
13	Pattern mining	
14	Outlier analysis	
15	Outlier analysis	Final Project proposal due
16	Classifiers	
17	Midterm	
18	Classifiers	
19	Optimization and model fitting	
20	Optimization and model fitting	
21	Causal modeling	
22	Streaming data	Final Project preliminary report due
23	Text data	
24	Time series data	
25	Graph data	
26	Scalability and data engineering	
27	Data journalism	
	Final project presentation	Final Project slides and final report due

Why Graph Mining

Graphs are everywhere

- chemical compounds (Cheminformatics)
- protein structures, biological pathways/networks (Bioinformactics)
- program control flow, traffic flow, and workflow analysis
- XML databases, Web, and social network analysis

Graph is a general model

- trees, lattices, sequences, and items are degenerated graphs

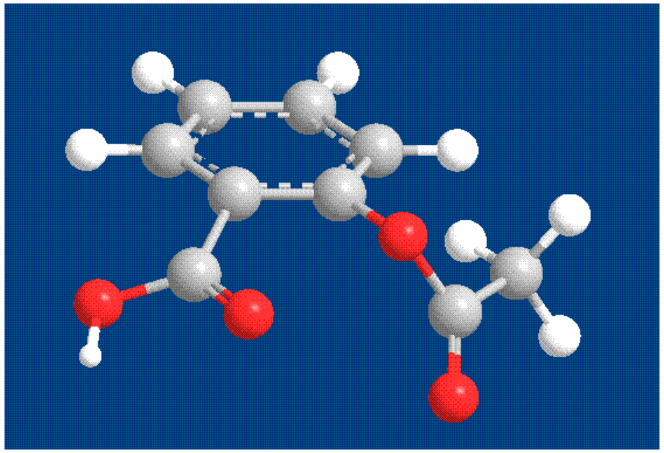
Diversity of graphs

- directed vs. undirected, labeled vs. unlabeled (edges & vertices), weighted, with angles & geometry (topological vs. 2-D/3-D)

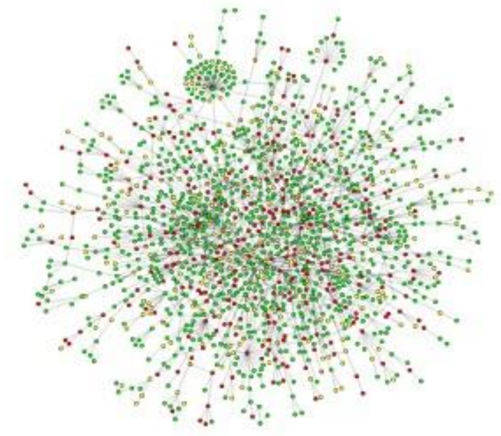
Complexity of algorithms:

- many problems are of high complexity (NP complete)

Graphs are Everywhere

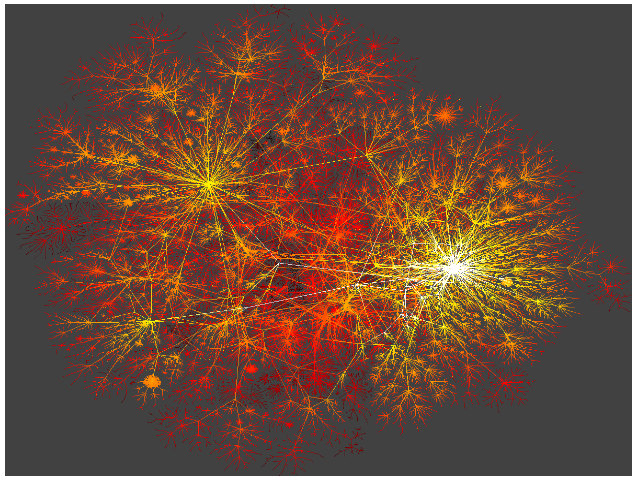


Aspirin

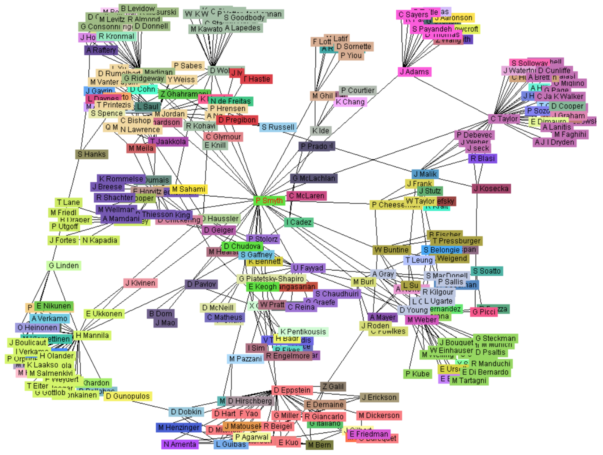


from H. Jeong et al Nature 411, 41 (2001)

Yeast protein interaction network

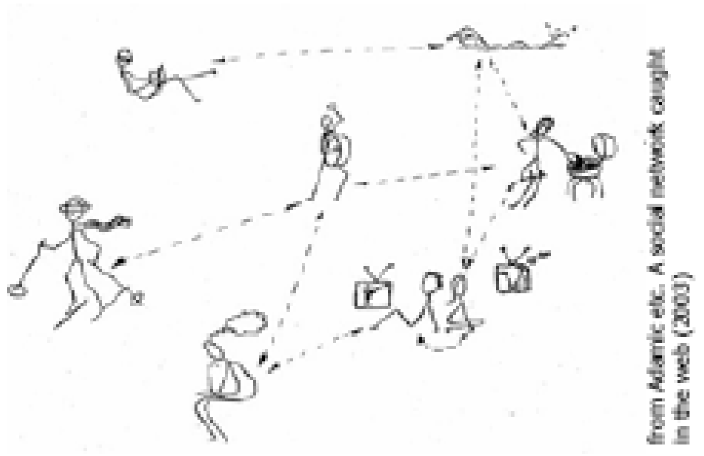


Internet

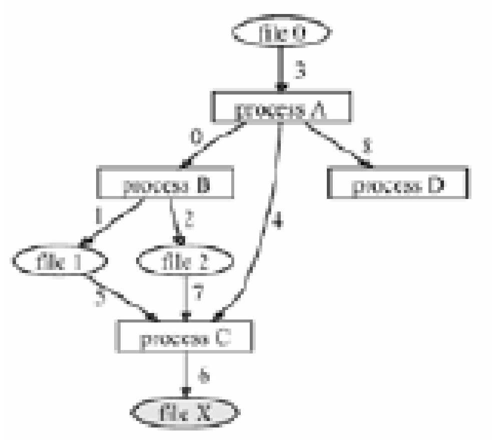


Co-author network

Graphs are Everywhere

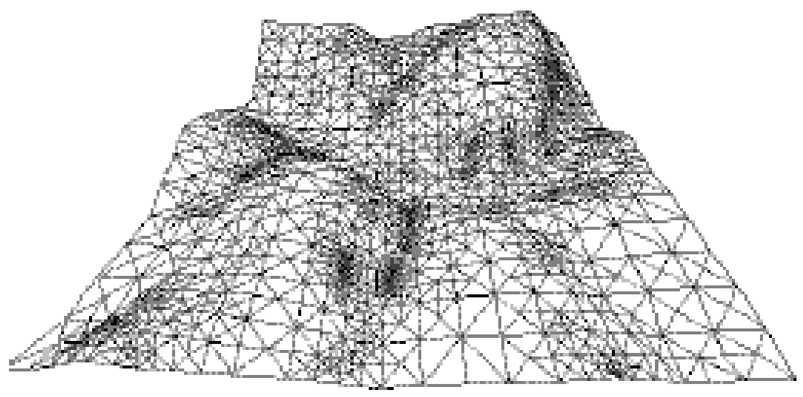

from Atlantic etc. A social network caught in the web (2003)

Social Network



Event Log Graph

Workflow



Mesh

Modeling Data With Graphs

Graphs are suitable for capturing arbitrary relations between the various elements.

Data Instance		**Graph Instance**
Element	⟺	Vertex
Element's Attributes	⟺	Vertex Label
Relation Between Two Elements	⟺	Edge
Type Of Relation	⟺	Edge Label
Relation between a Set of Elements	⟺	Hyper Edge

Provide enormous flexibility for modeling the underlying data as they allow the modeler to decide on what the elements should be and the type of relations to be modeled

Graph Pattern Mining

Frequent subgraphs

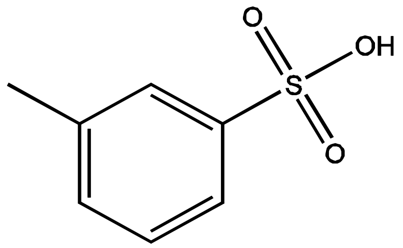
- a (sub)graph is ***frequent*** if its support (occurrence frequency) in a given dataset is no less than a minimum support threshold
- support of a graph g is defined as the percentage of graphs in G which have g as subgraph

Applications of graph pattern mining:

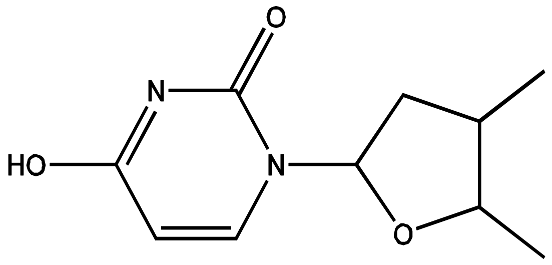
- mining biochemical structures
- program control flow analysis
- mining XML structures or Web communities
- building blocks for graph classification, clustering, compression, comparison, and correlation analysis

Example – Frequent Subgraphs

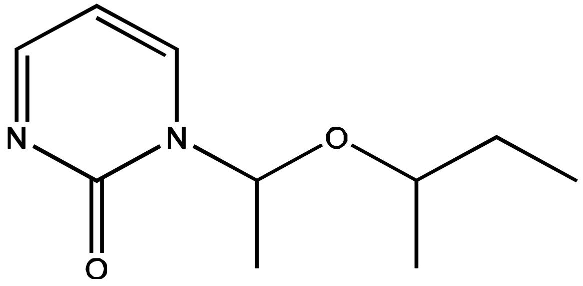
GRAPH DATASET



(T1)



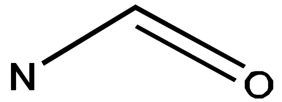
(T2)



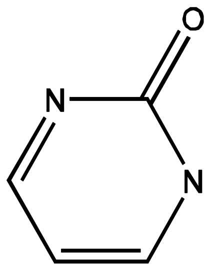
(T3)

FREQUENT PATTERNS
(MIN SUPPORT IS 2)

(1)

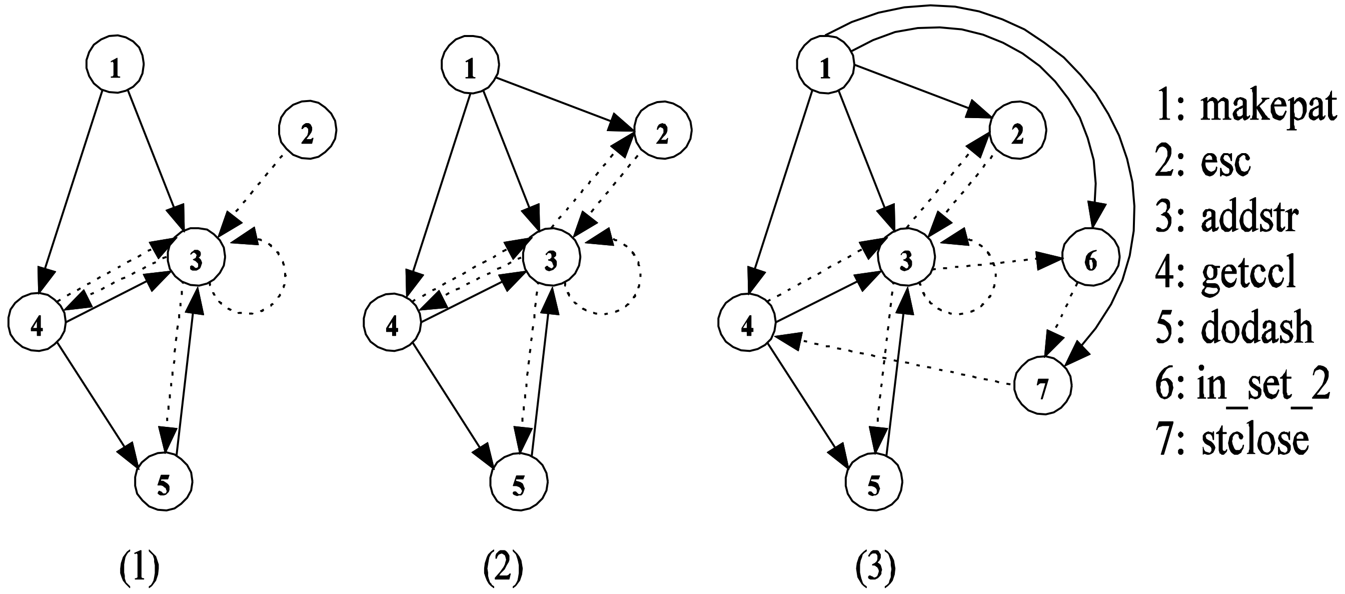


(2)

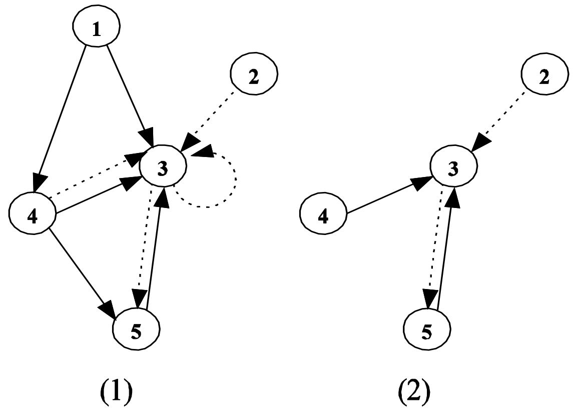


Example – Frequent Subgraphs

GRAPH DATASET

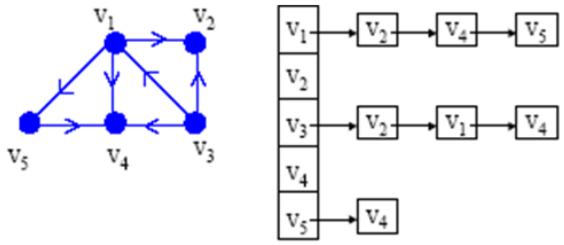


FREQUENT PATTERNS (MIN SUPPORT IS 2)

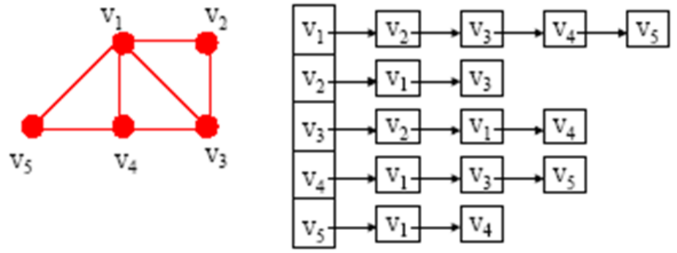


Graph Representations

- adjacency list

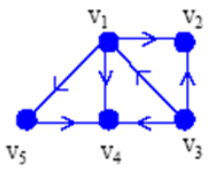


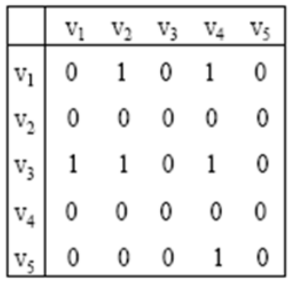
$v_1 \to v_2 \to v_4 \to v_5$
v_2
$v_3 \to v_2 \to v_1 \to v_4$
v_4
$v_5 \to v_4$



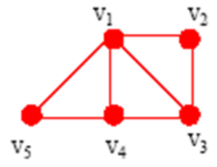
$v_1 \to v_2 \to v_3 \to v_4 \to v_5$
$v_2 \to v_1 \to v_3$
$v_3 \to v_2 \to v_1 \to v_4$
$v_4 \to v_1 \to v_3 \to v_5$
$v_5 \to v_1 \to v_4$

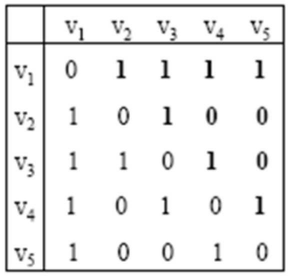
- adjacency matrix





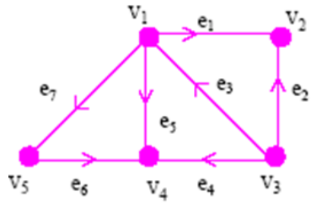
	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	0
v_2	0	0	0	0	0
v_3	1	1	0	1	0
v_4	0	0	0	0	0
v_5	0	0	0	1	0

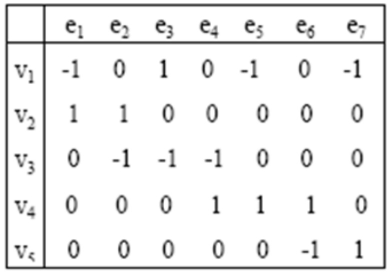




	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	1	1
v_2	1	0	1	0	0
v_3	1	1	0	1	0
v_4	1	0	1	0	1
v_5	1	0	0	1	0

- incidence matrix

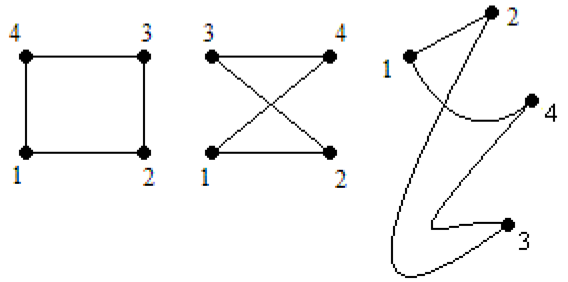




	e_1	e_2	e_3	e_4	e_5	e_6	e_7
v_1	-1	0	1	0	-1	0	-1
v_2	1	1	0	0	0	0	0
v_3	0	-1	-1	-1	0	0	0
v_4	0	0	0	1	1	1	0
v_5	0	0	0	0	0	-1	1

ISOMORPHISM

Formalizes the notion of equal graphs



More formally an isomorphism of graphs *G*1 and *G*2 is a bijection *f*:*V*(*G*1)↦*V*(*G*2) that preserves adjacency

If *G*1=*G*2 then the obtained mapping becomes an automorphism - a isomorphism from the graph to itself

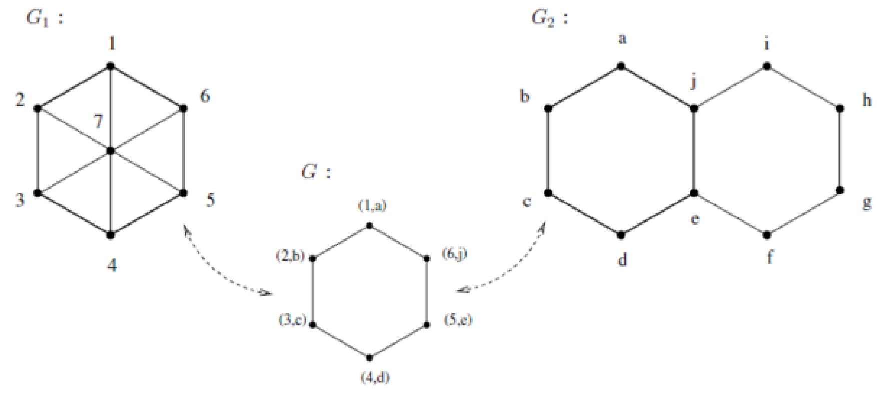
- if there is an automorphism of *f* of graph *G* such that the vertex *v* is mapped to vertex *u* then in a way the neighborhood of *u* and *v* "looks" the same

Subgraph Isomorphism

Two subgraphs that are isomorphic

Maximum common subgraph (MCS)

- the largest possible subgraph that cannot be extended by an addition of a vertex
- finding it is an optimization problem that is known to be NP-hard
- there can be many MCS for a pair of graphs G_1 and G_2



MCG-based Distances

When two graphs share a large subgraph in common, it is indicative of similarity

Un-normalized non-matching measure

$$U(G_1, G_2) = |G_1| + |G_2| - 2 \cdot |MCS(G_1, G_2)|$$

- equal to the number of non-matching nodes between the two graphs because it subtracts out the number of matching nodes |*MCS*(*G*1,*G*2)| from each of |*G*1| and |*G*2| and then sums them
- unnormalized because the value of the distance depends on the raw size of the underlying graphs.
- not desirable because it is more difficult to compare distances between pairs of graphs of varying size
- more effective when the different graphs in the collection are of approximately similar size.

MCG-based Distances

Union-normalized distance

- within [0.1]

$$UDist(G_1, G_2) = 1 - \frac{|MCS(G_1, G_2)|}{|G_1| + |G_2| - |MCS(G_1, G_2)|}$$

- normalizes the number of non-matching nodes U(G1,G2) between the two graphs (unnormalized measure) with the number of nodes in the union of the two graphs

$$UDist(G_1, G_2) = \frac{\text{Non-matching nodes between } G_1 \text{ and } G_2}{\text{Union size of } G_1 \text{ and } G_2}$$

- intuitively easier to interpret– two perfectly matching graphs will have a distance of 0 from one another, and two perfectly nonmatching
- graphs will have a distance of 1

MCG-based Distances

Max-normalized distance

- within [0.1]

$$MDist(G_1, G_2) = 1 - \frac{|MCS(G_1, G_2)|}{\max\{|G_1|, |G_2|\}}$$

Any of these distance measures can be computed effectively only for small graphs

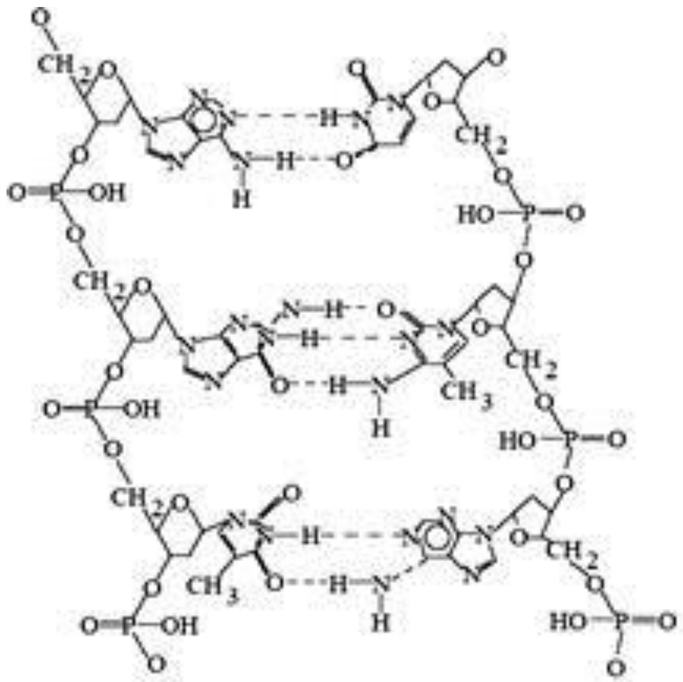
- for larger graphs, it becomes computationally too expensive to evaluate these measures because of the need to determine the maximum common subgraph between the two graphs
- use lexicon-base metrics for large graphs (see next)

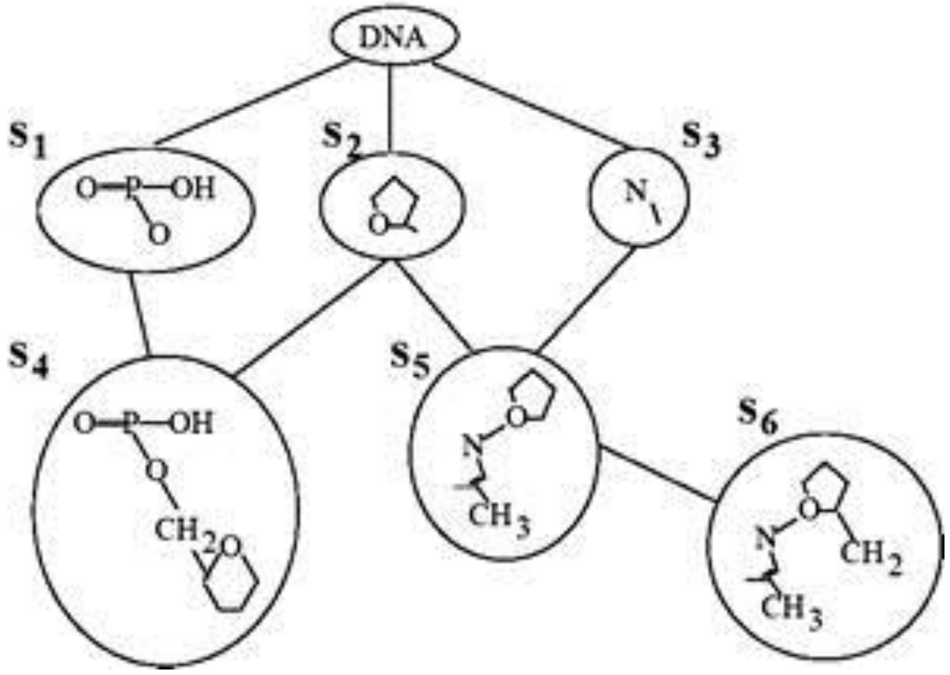
Frequent Substructure-Based Distance Computation

Algorithm

- create a lexicon of frequent subgraph patterns by frequent subgraph mining
- reduce the overlap among the frequent subgraph patterns
- create a new feature f_i for each frequent subgraph S_i selected
- it will create a feature set of size d
- for each graph G_i, create a vector-space representation in terms of the features $f_1 \ldots f_d$
- each graph contains the features, corresponding to the subgraphs that it contains
- the frequency of each feature is the number of occurrences of the corresponding subgraph in the graph G_i
- optionally apply tf-idf normalization
- use any similarity function to compute distances between graph objects

Example





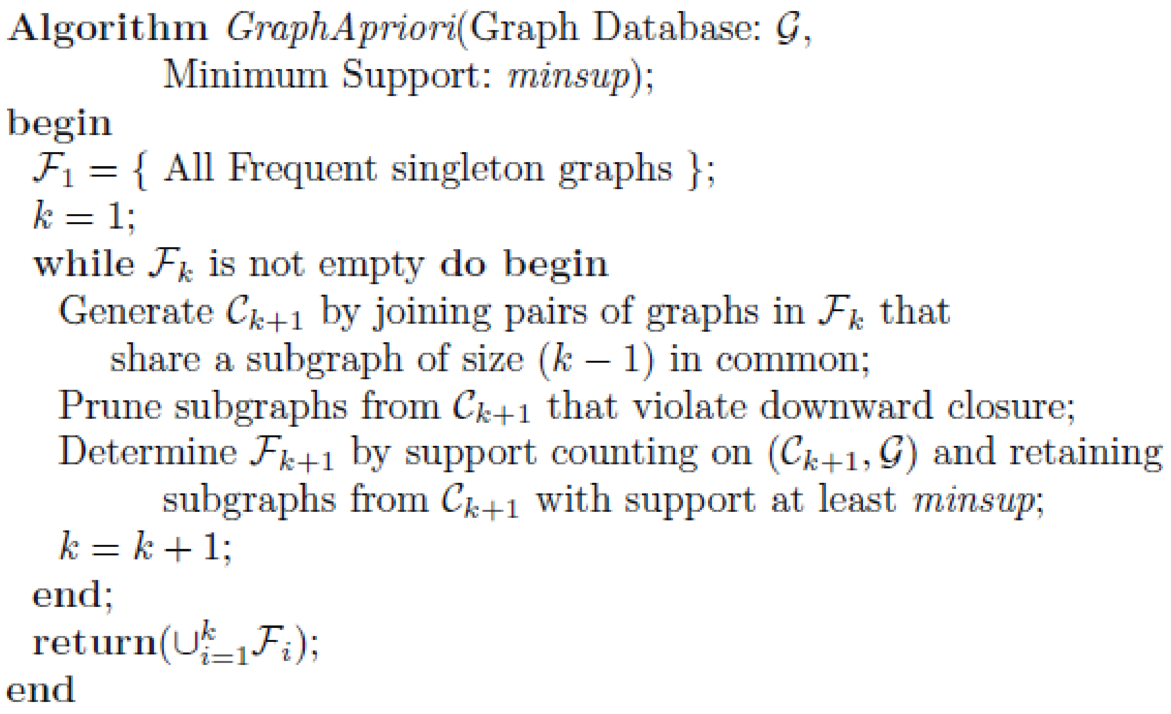
A-Priori Based Graph Mining

Looks for frequent sub-structures (sub-graphs)

Recall a-priori pruning principle:

- if there is any item set which is infrequent, then its superset should not be generated/tested
- apply the same principle for sub-graphs
- size of a subgraph may refer to either its nodes or edges depending on the specific algorithm used

A-Priori Based Approach

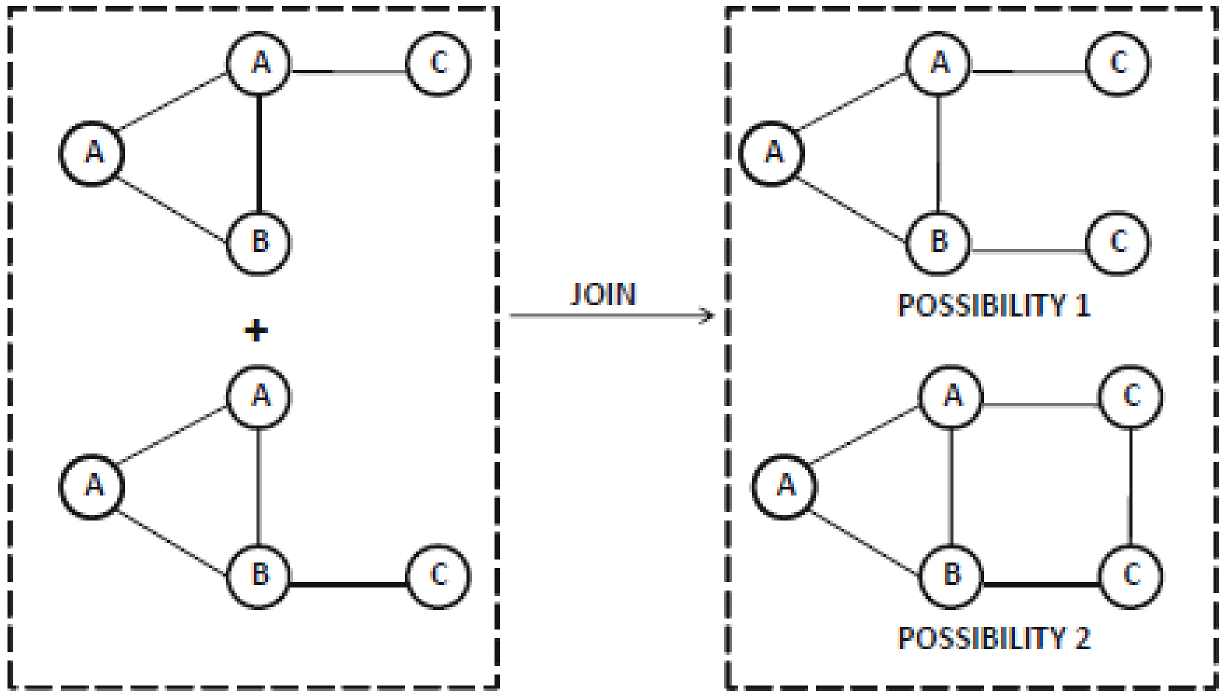


```
Algorithm GraphApriori(Graph Database: $\mathcal{G}$,
          Minimum Support: minsup);
begin
  $\mathcal{F}_1$ = { All Frequent singleton graphs };
  $k = 1$;
  while $\mathcal{F}_k$ is not empty do begin
    Generate $\mathcal{C}_{k+1}$ by joining pairs of graphs in $\mathcal{F}_k$ that
        share a subgraph of size $(k-1)$ in common;
    Prune subgraphs from $\mathcal{C}_{k+1}$ that violate downward closure;
    Determine $\mathcal{F}_{k+1}$ by support counting on $(\mathcal{C}_{k+1}, \mathcal{G})$ and retaining
          subgraphs from $\mathcal{C}_{k+1}$ with support at least minsup;
    $k = k + 1$;
  end;
  return($\cup_{i=1}^{k} \mathcal{F}_i$);
end
```

Downward closure

- any subset of a frequent itemset must be frequent

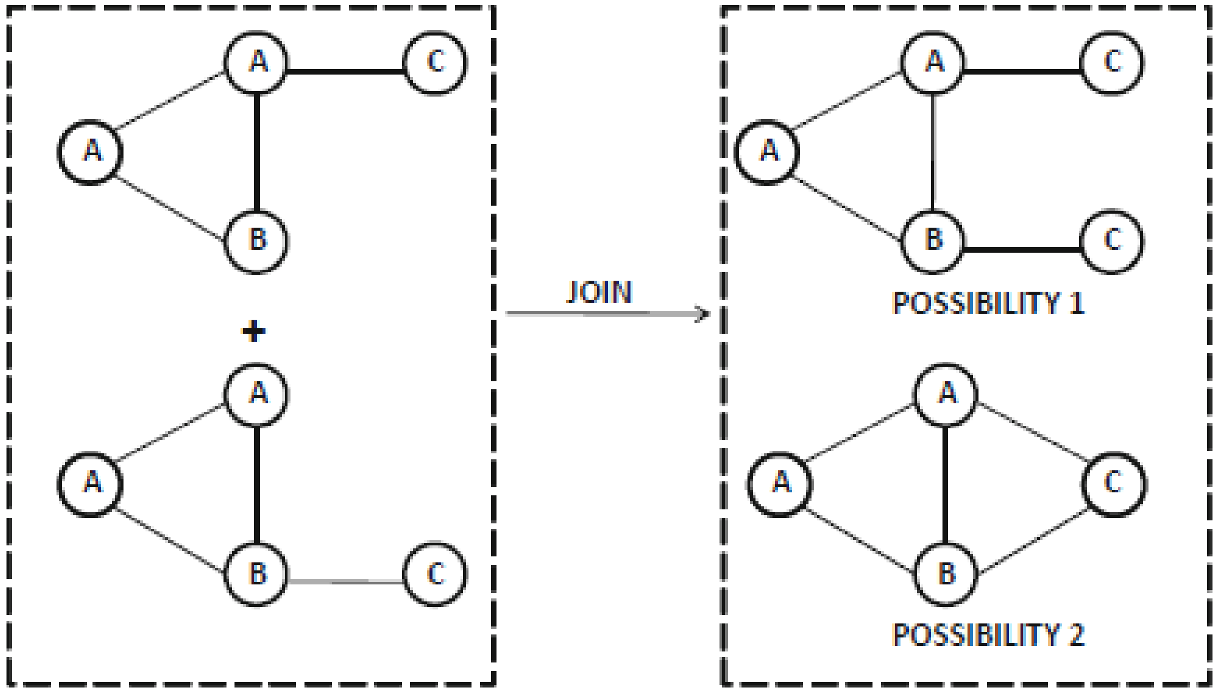
Example: Node-Based Join



Number of edges is not constrained

- this can lead to ambiguities

Example: Edge-Based Join



Number of nodes is not constrained

- new level could have the same number than next level
- another form of ambiguity

Graph Clustering

Either distance-based or frequent substructure-based

- distance-based methods are more effective for smaller graphs, in which distances can be computed robustly and efficiently
- frequent substructure-based methods are appropriate for larger graphs where distance computations become qualitatively and computationally impractical

Distance-based clustering

- use methods like k-medoids or spectral clustering
- computationally expensive to compute distances between large graph objects
- effectiveness also suffers for large graphs because these graphs may be similar only in some portions that repeat frequently

  → the rare (and unique) portions of the graph may not factor in
- might use a substructure-based distance function instead

Graph Clustering

Frequent substructure-based methods

- extract frequent subgraphs from the data and use their membership in input graphs to determine clusters

Algorithm

- apply frequent subgraph mining methods discussed to discover frequent subgraph patterns in the underlying graphs
- select a subset of subgraphs to reduce overlap among the different subgraphs
- create a new feature f_i for each frequent subgraph S_i discovered
- gives rise to lexicon of d features
- create a d-vector of features – one element per feature
- represent each graph in terms of the features and their frequency
- cluster as usual

The XProj Algorithm

A frequent substructure-based clustering method

Originally proposed for XML graphs

- a substructure can be viewed as a PROJection of the graph

```
Algorithm XProj(Graph Database: $\mathcal{G}$, Minimum Support: $minsup$
        Structural Size: $l$, Number of Clusters: $k$ )
begin
  Initialize clusters $\mathcal{C}_1 \dots \mathcal{C}_k$ randomly;
  Compute frequent substructure sets $\mathcal{F}_1 \dots \mathcal{F}_k$ from $\mathcal{C}_1 \dots \mathcal{C}_k$;
  repeat
    Assign each graph $G_j \in \mathcal{G}$ to the cluster $\mathcal{C}_i$ for which the former's
      similarity to $\mathcal{F}_i$ is the largest $\forall i \in \{1 \dots k\}$;
    Compute frequent substructure set $\mathcal{F}_i$ from $\mathcal{C}_i$ for each $i \in \{1 \dots k\}$;
  until convergence;
end
```

Graph Classification

Use either distance or pattern/structure based methods discussed before to form the decision metric for any standard classifier

- kernel-based methods are also possible