CS6220: DATA MINING TECHNIQUES

Mining Graph/Network Data: Part I

Instructor: Yizhou Sun

yzsun@ccs.neu.edu

November 12, 2013

Announcement

- Homework 4 will be out tonight
    - Due on 12/2
- Next class will be canceled
    - I will still put the last set of slides online, you can learn it by yourself
    - I will be in office next Tuesday afternoon (2-5pm), as the Wednesday office hour is in holiday
- Course project
    - Everyone is required to attend both sessions (12/3 and 12/10)
    - Presentation will be increased to 15 mins / group, as we now have two sessions
    - More details will be announced in Piazza

New course next semester

- Spring 2014, [CS 7280 Special Topics in Data Mining (Mining Information/Social Networks)]
  - Paper reading and presentation (20%)
  - Homework (20%)
  - Research project (50%)
  - Participation (10%)

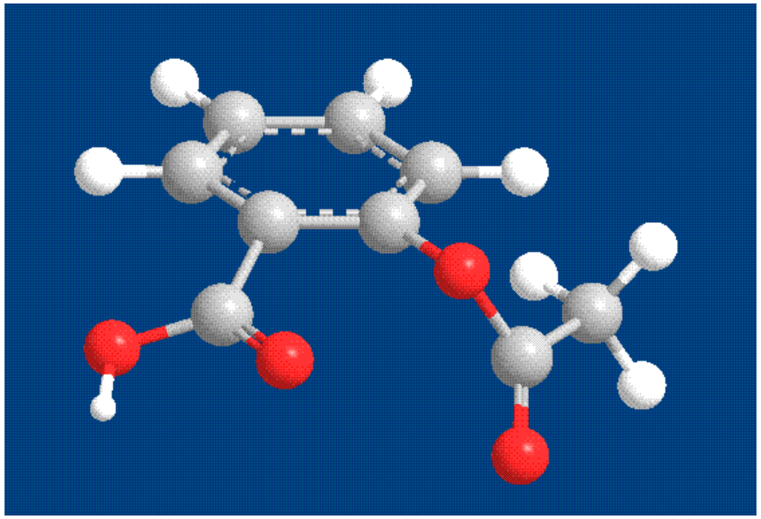
Tentative Syllabus

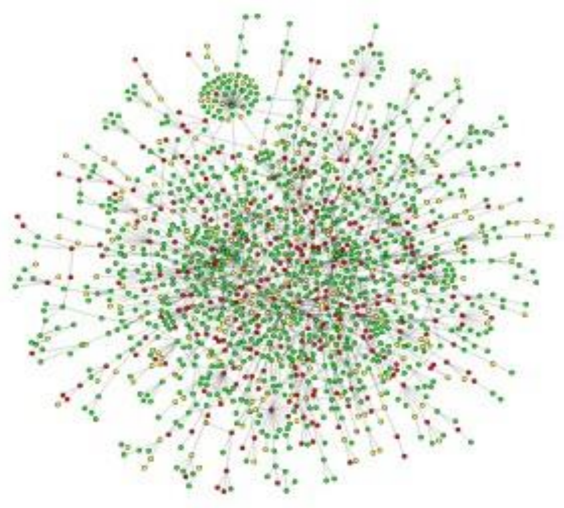
- 1. Basics of Information/Social Networks
  2. Ranking for infonet
  3. Clustering / community detection
  4. Matrix factorization
  5. Classification / label propagation / node or link profiling
  6. Probabilistic models for infonets
  7. Similarity search
  8. Diffusion / Influence maximization
  9. Recommendation
  10. Link / relationship prediction
  11. Trustworthy analysis
  12. Large graph computation
  13. Network evolution

Mining Graph/Network Data: Part I

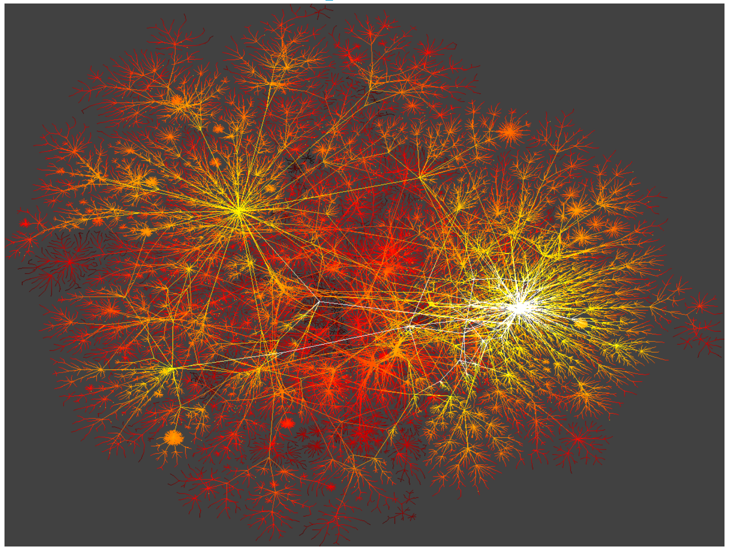
- Graph / Network Data
- Graph Pattern Mining
- Ranking on Graph / Network
- Summary

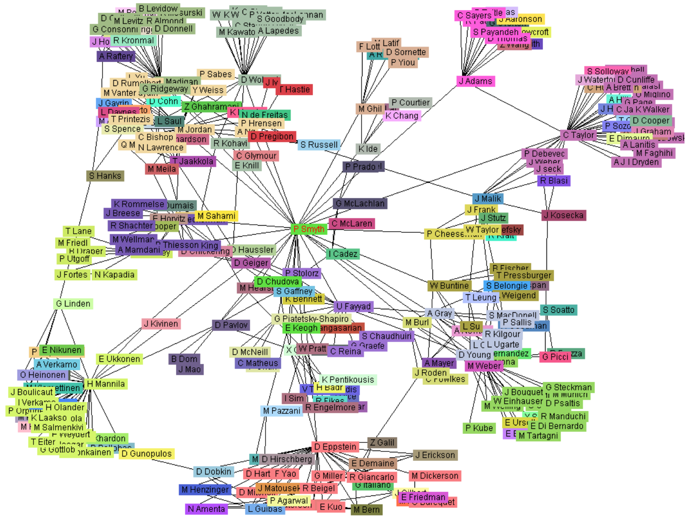
Graph, Graph, Everywhere


Aspirin


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

Yeast protein interaction network


Internet


Co-author network

Why Graph Mining?

- Graphs are ubiquitous
  - 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

Representation of a Graph

- $G =< V, E >$
  - $V = \{u_1, \dots, u_n\}$: node set
  - $E \subseteq V \times V$: edge set
- Adjacency matrix
  - $A = \{a_{ij}\}, i, j = 1, \dots, n$
    - $a_{ij} = 1, if \ < u_i, u_j > \in E$
    - $a_{ij} = 0, if \ < u_i, u_j > \notin E$
  - Undirected graph vs. Directed graph
    - $A = A^{\mathrm{T}} \; vs. A \neq A^{\mathrm{T}}$
  - Weighted graph
    - Use *W* instead of *A*, where w_{ij} represents the weight of edge $< u_i, u_j >$

Mining Graph/Network Data: Part I

- Graph / Network Data
- Graph Pattern Mining
- Ranking on Graph / Network
- Summary

Graph Pattern Mining

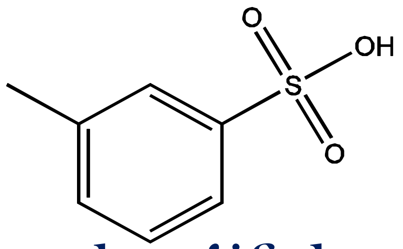
- Mining Frequent Subgraph Patterns
- Graph Search

Mining Frequent Subgraph Patterns

- *Frequent* subgraphs
  - A (sub)graph is ***frequent*** if its *support* (occurrence frequency) in a given dataset is no less than a *minimum support* threshold
- 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

Labeled Graph and Subgraph

- Labeled graph
  - A label function maps each vertex or edge to a label
    - E.g., a molecule is a labeled graph



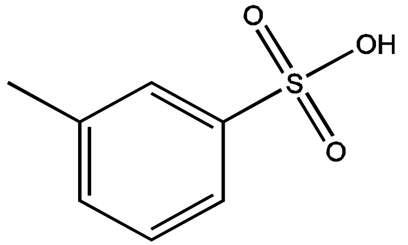
- Subgraph
  - A graph *g* is a subgraph of another graph *g'* if there exists a subgraph isomorphism from *g* to *g'*
    - *There exists a subgraph* $g_0' \subseteq g'$, *such that g is graph isomorphism to* g_0', *i.e., there is a bijective mapping between nodes in g and* g_0', *such that for every edge in g, the mapped node pair is also an edge in* g_0'
    - *For labeled graph, we also required the labels after the mapping are the same*

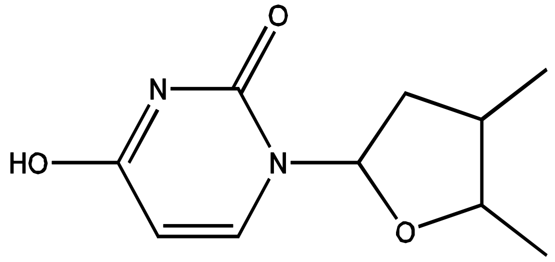
Support of a Subgraph

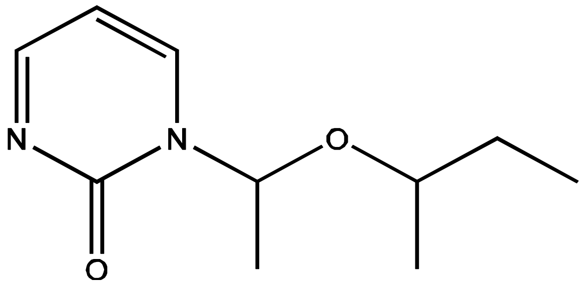
- Given a graph database
  - $D = \{G_1, \dots, G_n\}$
- The support of a graph g, support(g), is:
  - The number of graphs in the database that g is a subgraph
- Frequent graph
  - A graph whose support is equal or larger than min_sup

Example: Frequent Subgraphs

GRAPH DATASET



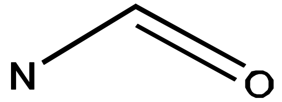




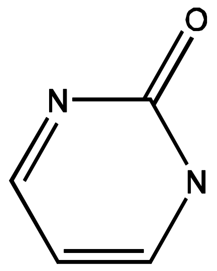
(A) (B) (C)

FREQUENT PATTERNS (MIN SUPPORT IS 2)

(1)

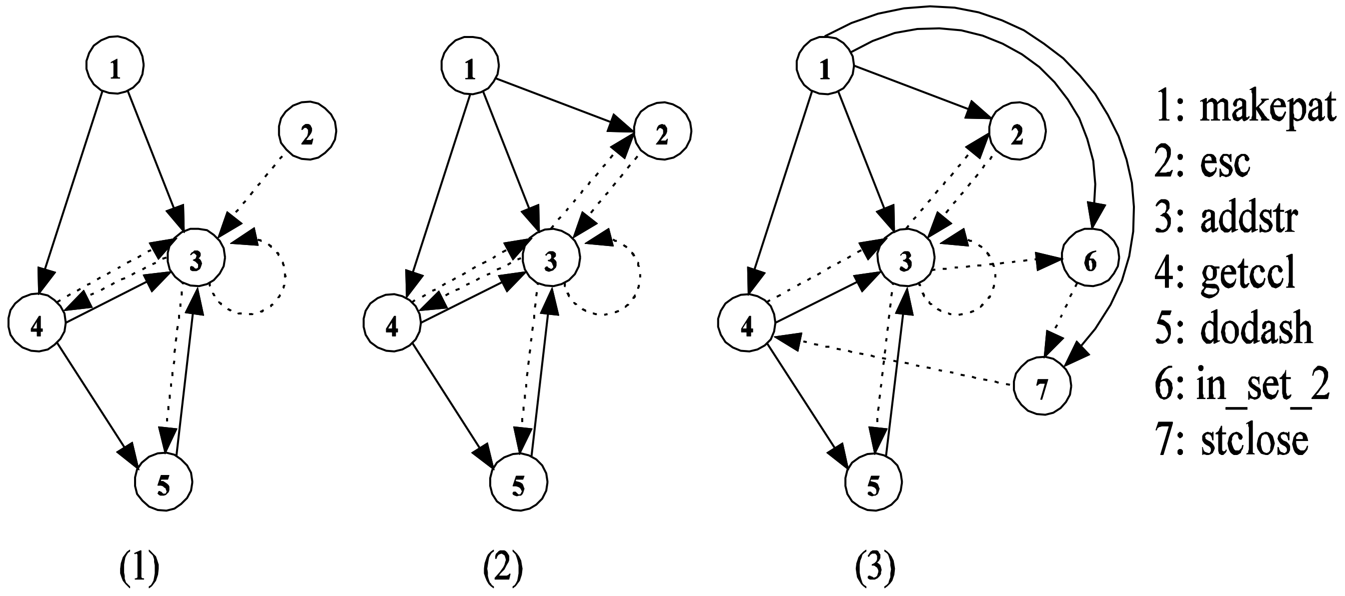


(2)

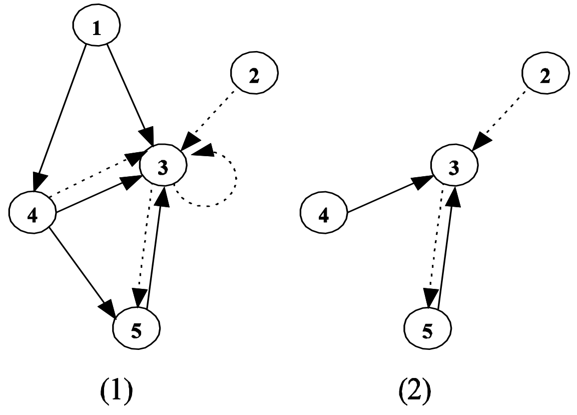


EXAMPLE (II)

GRAPH DATASET



FREQUENT PATTERNS (MIN SUPPORT IS 2)



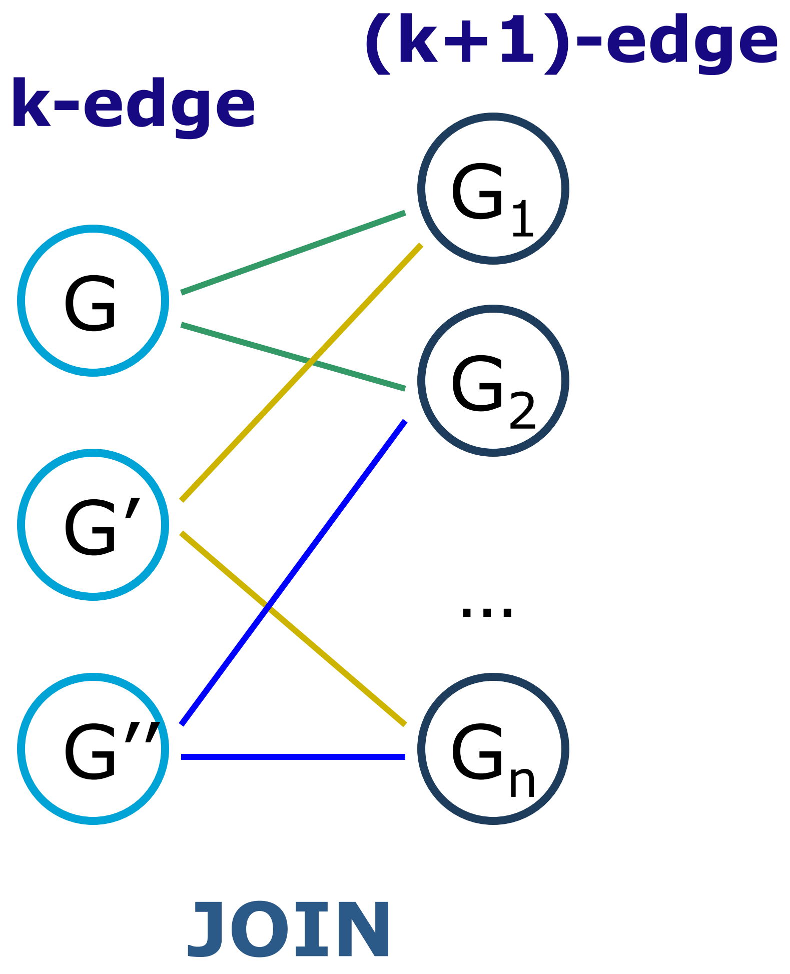
How to Mine Frequent Subgraph Pattern?

- Two steps
  - Step 1: Generate frequent substructure candidates
  - Step 2: Calculate the support of these candidates using subgraph isomorphism test (NP!)
- Two types of approaches
  - Apriori-based approach
  - Pattern-growth approach

Frequent Subgraph Mining Approaches

- Apriori-based approach
  - AGM/AcGM: Inokuchi, et al. (PKDD'00)
  - FSG: Kuramochi and Karypis (ICDM'01)
  - PATH#: Vanetik and Gudes (ICDM'02, ICDM'04)
  - FFSM: Huan, et al. (ICDM'03)
- Pattern growth approach
  - MoFa, Borgelt and Berthold (ICDM'02)
  - gSpan: Yan and Han (ICDM'02)
  - Gaston: Nijssen and Kok (KDD'04)

Apriori-Based Approach



Apriori Approach Framework

Algorithm: AprioriGraph. Apriori-based frequent substructure mining.

Input:

- D, a graph data set;
- min_sup, the minimum support threshold.

Output:

- S_k, the frequent substructure set.

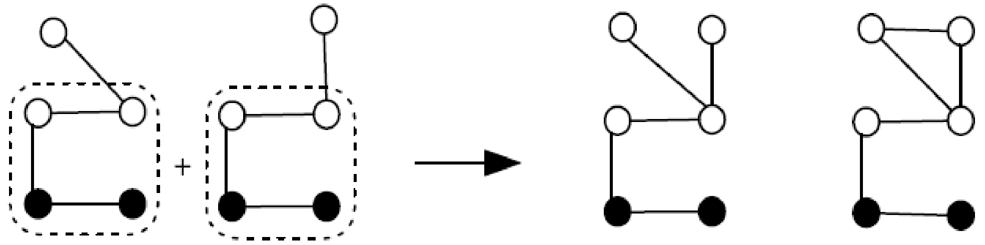
Method:

```
$S_1 \leftarrow$ frequent single-elements in the data set;
Call AprioriGraph($D$, $min\_sup$, $S_1$);

procedure AprioriGraph($D$, $min\_sup$, $S_k$)

(1) $S_{k+1} \leftarrow \varnothing$;
(2) for each frequent $g_i \in S_k$ do
(3)     for each frequent $g_j \in S_k$ do
(4)         for each size $(k+1)$ graph $g$ formed by the merge of $g_i$ and $g_j$ do
(5)             if $g$ is frequent in $D$ and $g \notin S_{k+1}$ then
(6)                 insert $g$ into $S_{k+1}$;
(7) if $s_{k+1} \neq \varnothing$ then
(8)     AprioriGraph($D$, $min\_sup$, $S_{k+1}$);
(9) return;
```

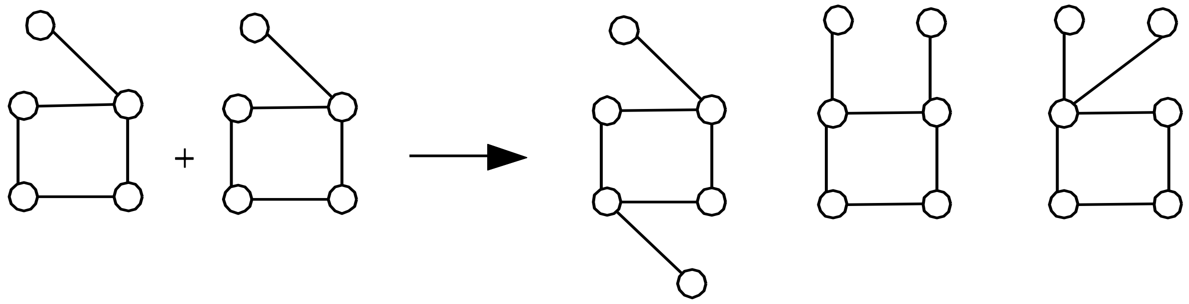
Apriori-Based, Breadth-First Search

- Methodology: breadth-search, joining two graphs

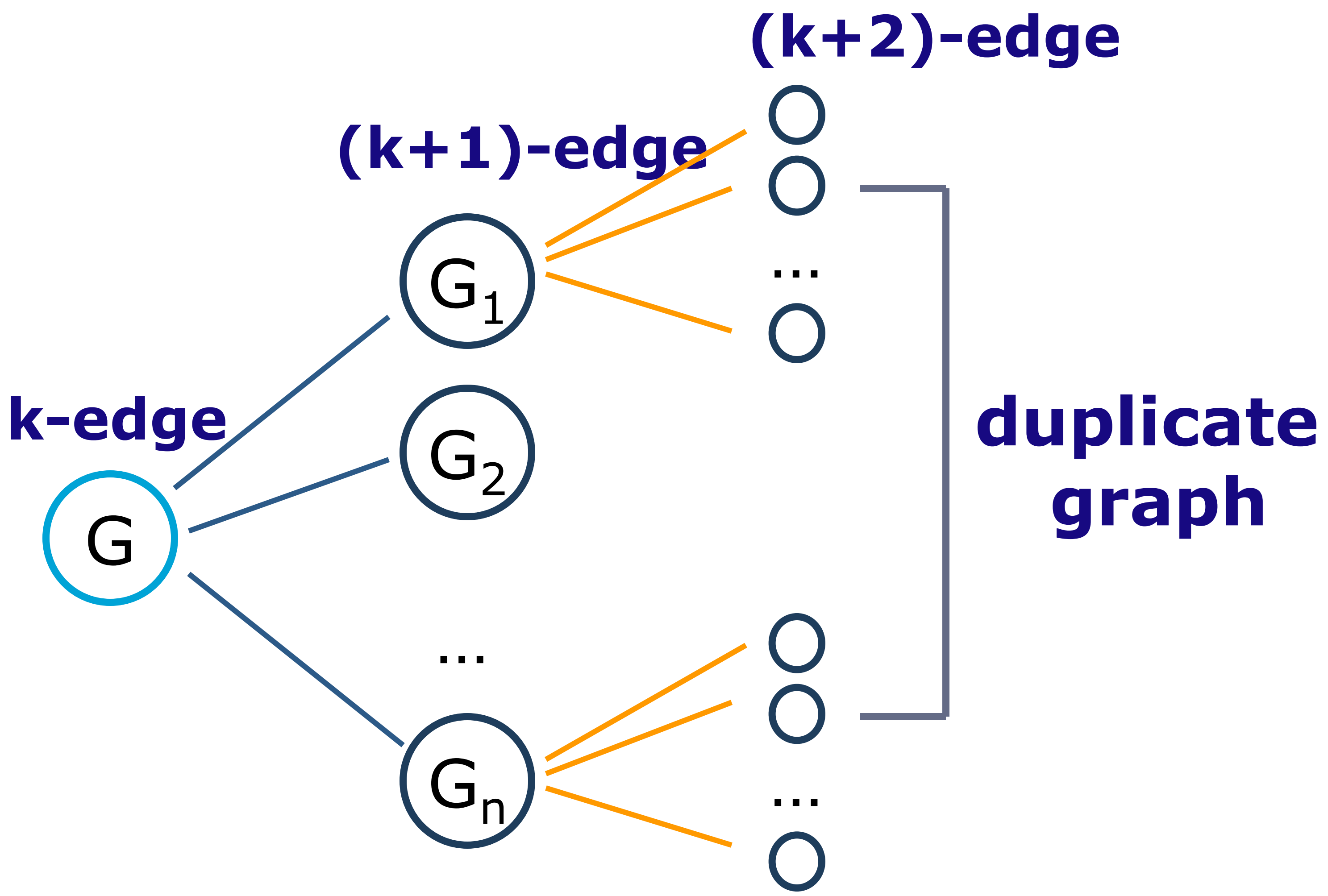
- AGM (Inokuchi, et al. PKDD'00)
  - generates new graphs with one more node



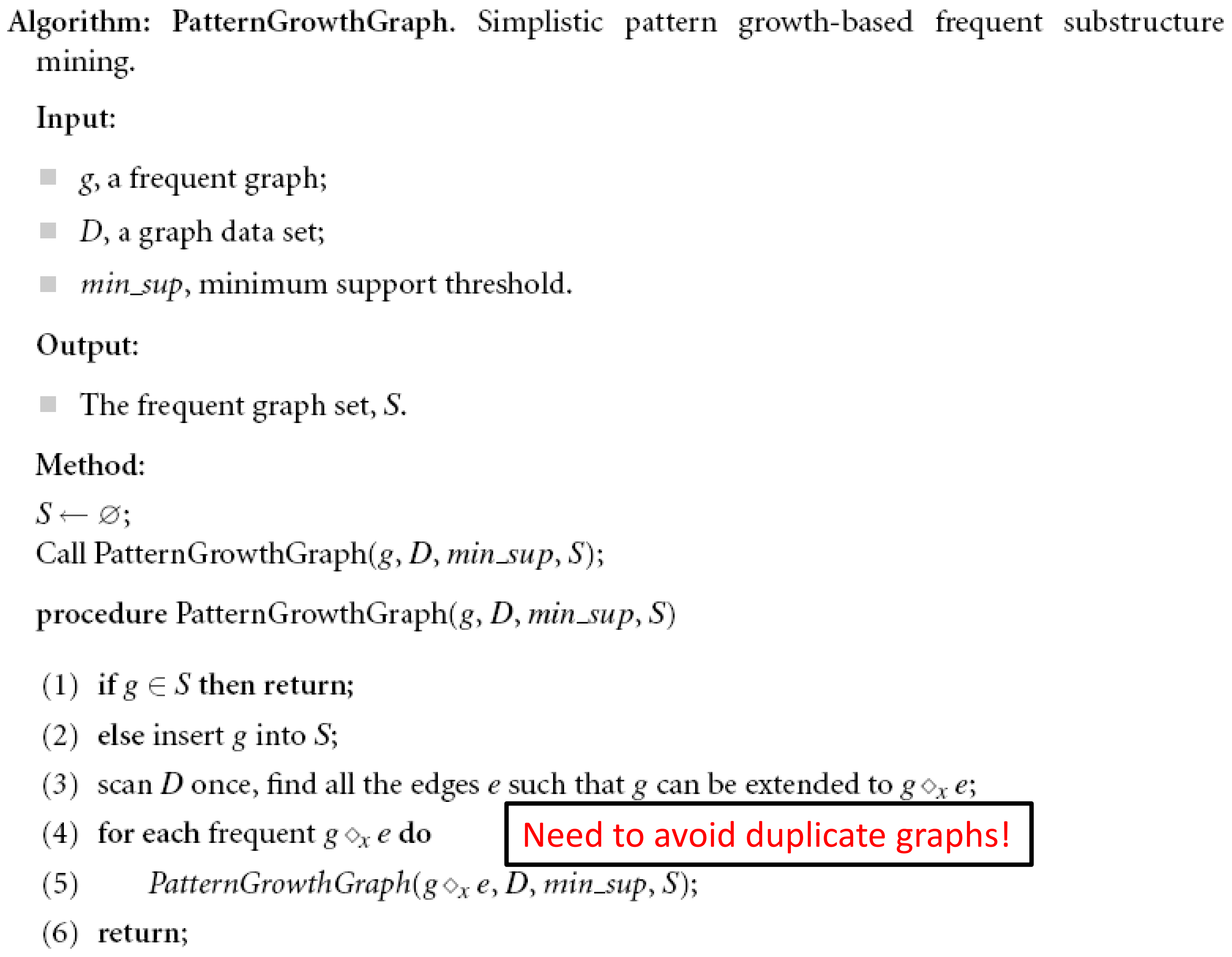
- FSG (Kuramochi and Karypis ICDM'01)
  - generates new graphs with one more edge



Pattern Growth Method



Pattern Growth Approach Framework



Algorithm: PatternGrowthGraph. Simplistic pattern growth-based frequent substructure mining.

Input:

- g, a frequent graph;
- D, a graph data set;
- min_sup, minimum support threshold.

Output:

- The frequent graph set, S.

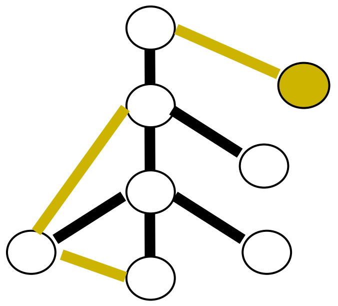
Method:

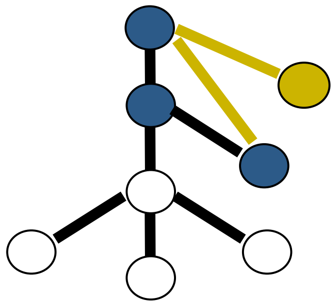
```
$S \leftarrow \varnothing$;
Call PatternGrowthGraph($g, D, min\_sup, S$);

procedure PatternGrowthGraph($g, D, min\_sup, S$)

(1) if $g \in S$ then return;
(2) else insert $g$ into $S$;
(3) scan $D$ once, find all the edges $e$ such that $g$ can be extended to $g \diamond_x e$;
(4) for each frequent $g \diamond_x e$ do
(5)         PatternGrowthGraph($g \diamond_x e, D, min\_sup, S$);
(6) return;
```

Need to avoid duplicate graphs!

GSPAN (Yan and Han ICDM'02)

Right-Most Extension



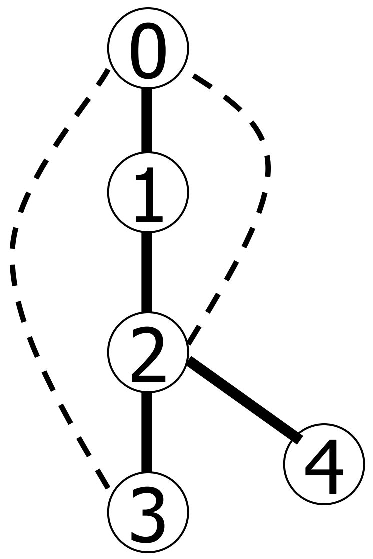


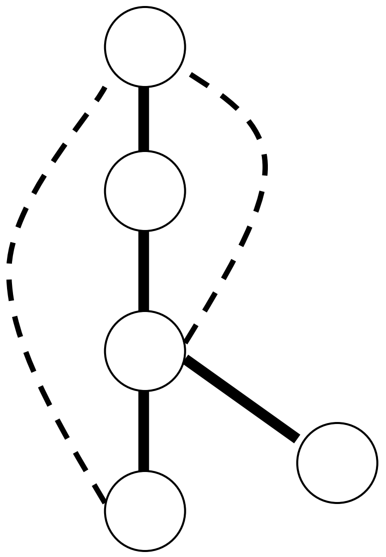
Theorem: Completeness

The Enumeration of Graphs using Right-most Extension is COMPLETE

DFS Code

- Flatten a graph into a sequence using depth first search





e0: (0,1)

e1: (1,2)

e2: (2,0)

e3: (2,3)

e4: (3,1)

e5: (2,4)

*DFS Lexicographic Order

- Let **Z** be the set of DFS codes of all graphs. Two DFS codes **a** and **b** have the relation **a<=b** (DFS Lexicographic Order in Z) if and only if one of the following conditions is true. Let

  a = $(x_0, x_1, \ldots, x_n)$ and

  b = $(y_0, y_1, \ldots, y_n)$,

(i) if there exists t, $0 <= t <= \min(m,n)$, $x_k = y_k$ for all k, s.t. $k<t$, and $x_t < y_t$

(ii) $x_k = y_k$ for all k, s.t. $0 <= k <= m$ and $m <= n$.

*DFS Code Extension

- Let **a** be the minimum DFS code of a graph **G** and **b** be a non-minimum DFS code of **G**. For any DFS code **d** generated from **b** by one right-most extension,

  (i) **d** is not a minimum DFS code,

  (ii) min_dfs(**d**) cannot be extended from **b**, and

  (iii) min_dfs(**d**) is either less than **a** or can be extended from **a**.

THEOREM [RIGHT-EXTENSION]
The DFS code of a graph extended from a Non-minimum DFS code is NOT MINIMUM

Graph Pattern Explosion Problem

- If a graph is frequent, all of its subgraphs are frequent – **the Apriori property**
- An **n**-edge frequent graph may have 2^n subgraphs
- Among **422** chemical compounds which are confirmed to be active in an AIDS antiviral screen dataset, there are **1,000,000** frequent graph patterns if the minimum support is 5%
  - To mine closed graph pattern directly
    - *CLOSEGRAPH (Yan & Han, KDD'03)

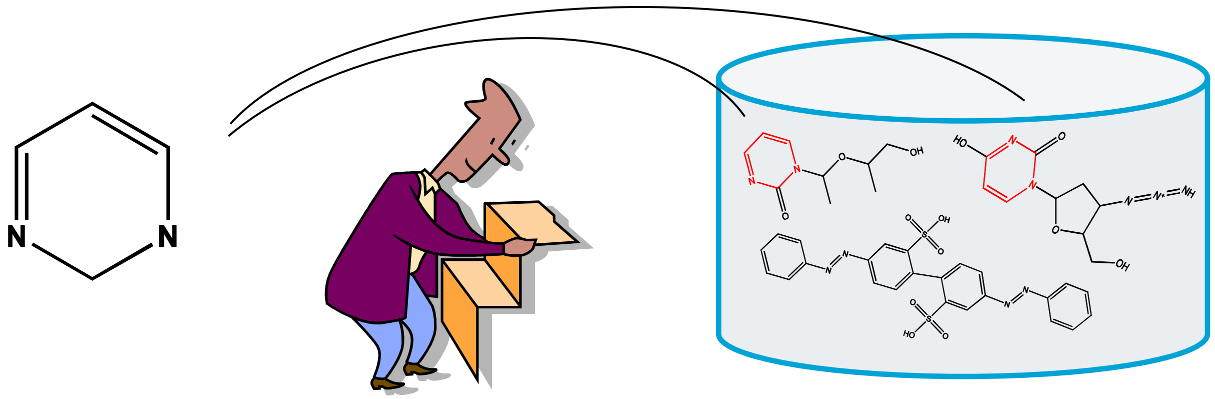
Graph Pattern Mining

- Mining Frequent Subgraph Patterns

- Graph Search

Graph Search

- Querying graph databases:
  - Given a graph database and a query graph, find all the graphs containing this query graph



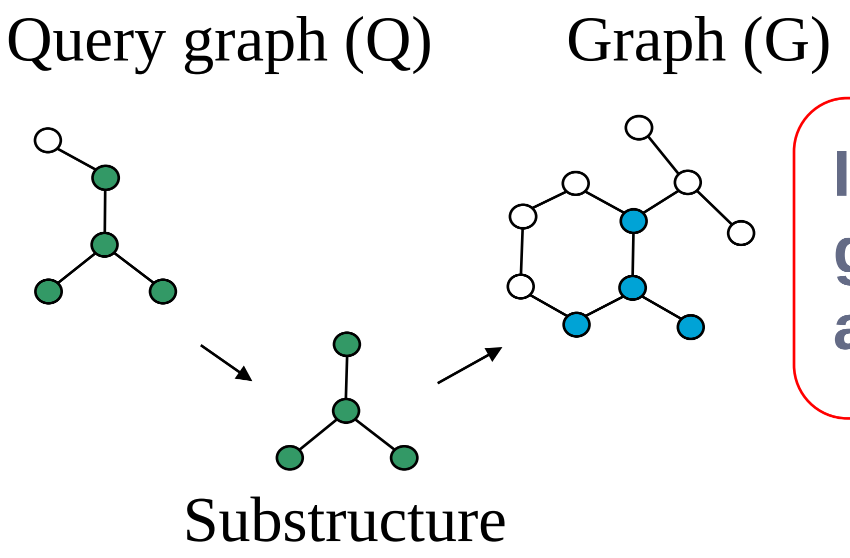
query graph

graph database

Scalability Issue

- Sequential scan
  - Disk I/Os
  - Subgraph isomorphism testing
- An indexing mechanism is needed
  - DayLight: Daylight.com (commercial)
  - GraphGrep: Dennis Shasha, et al. PODS'02
  - Grace: Srinath Srinivasa, et al. ICDE'03

Indexing Strategy



If graph G contains query graph Q, G should contain any substructure of Q

Remarks

- Index substructures of a query graph to prune graphs that do not contain these substructures

Indexing Framework

- Two steps in processing graph queries

Step 1. Index Construction

- Enumerate structures in the graph database, build an inverted index between structures and graphs

Step 2. Query Processing

- Enumerate structures in the query graph
- Calculate the candidate graphs containing these structures
- Prune the false positive answers by performing subgraph isomorphism test

Cost Analysis

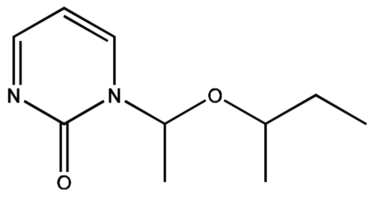
QUERY RESPONSE TIME

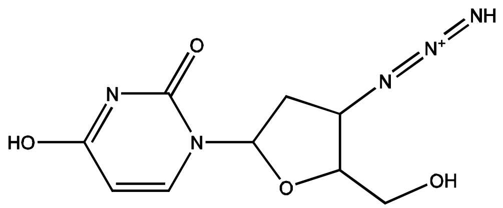
$$\underbrace{T_{index}}_{\text{fetch index}} + \underbrace{\boxed{\left|C_q\right|}}_{\text{number of candidates}} \times \left(T_{io} + T_{isomorphism\_testing}\right)$$

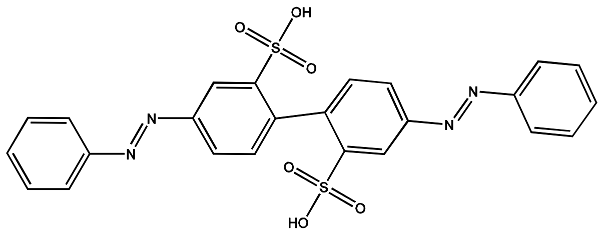
REMARK: make $|C_q|$ as small as possible

Path-based Approach

GRAPH DATABASE


(a)


(b)


(c)

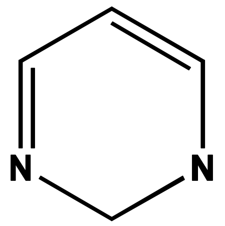
PATHS

0-length: C, O, N, S
1-length: C-C, C-O, C-N, C-S, N-N, S-O
2-length: C-C-C, C-O-C, C-N-C, ...
3-length: ...

Built an inverted index between paths and graphs

Path-based Approach (cont.)

QUERY GRAPH

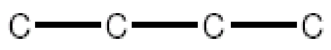


0-edge: S_C={a, b, c}, S_N={a, b, c}
1-edge: $S_{C\text{-}C}$={a, b, c}, $S_{C\text{-}N}$={a, b, c}
2-edge: $S_{C\text{-}N\text{-}C}$ = {a, b}, …
…

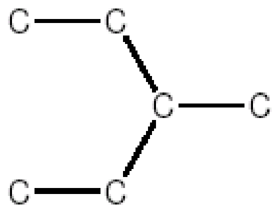
Intersect these sets, we obtain the candidate answers - graph (a) and graph (b) - which may contain this query graph.

Problems: Path-based Approach

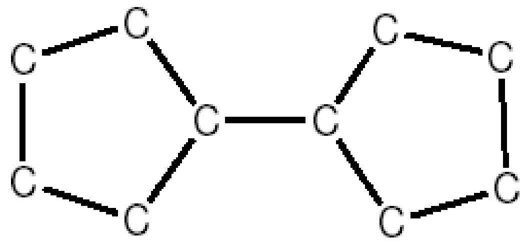
GRAPH DATABASE



(a)

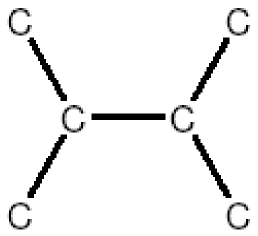


(b)



(c)

QUERY GRAPH

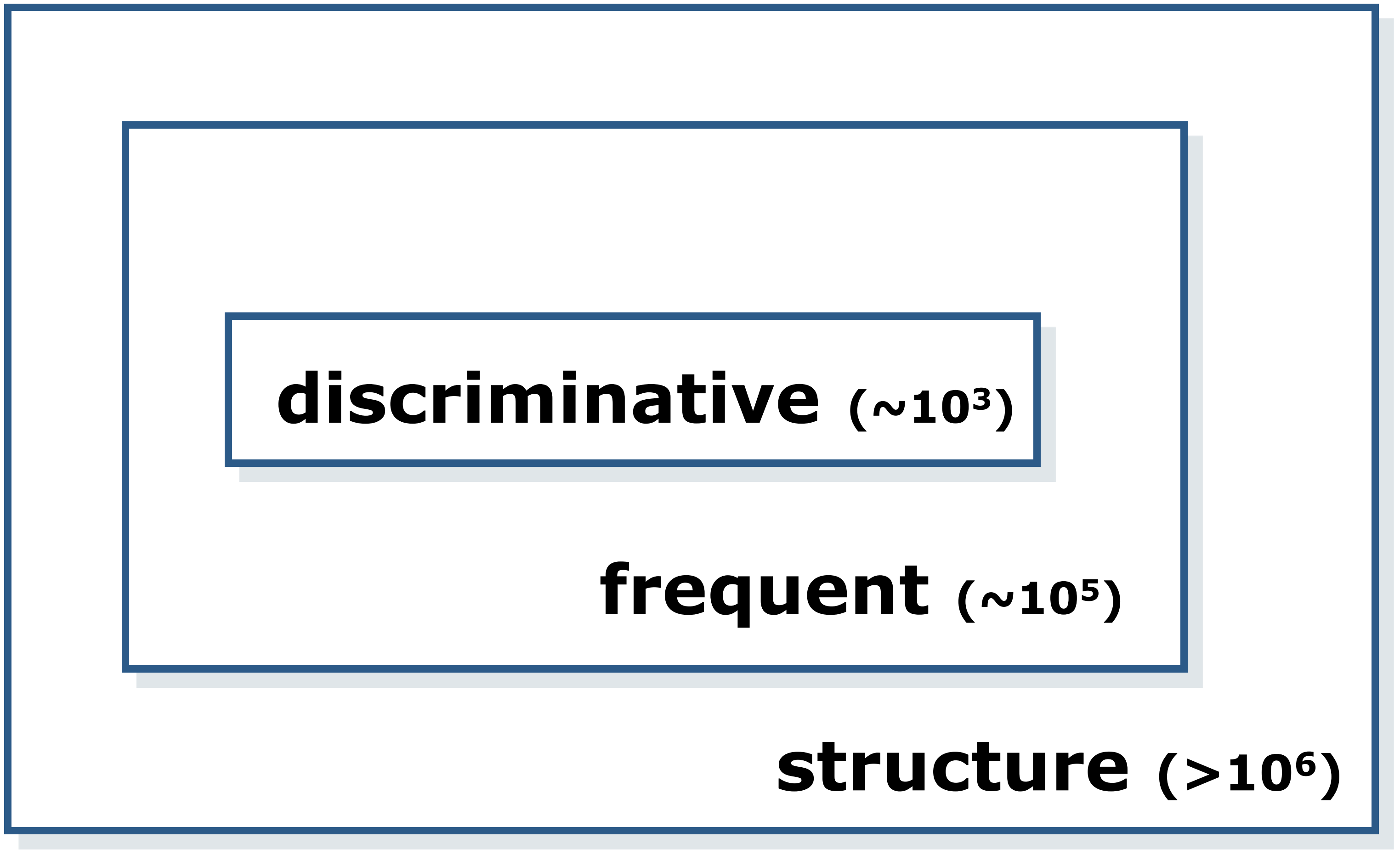


Only graph (c) contains this query graph. However, if we only index paths: C, C-C, C-C-C, C-C-C-C, we cannot prune graph (a) and (b).

gIndex: Indexing Graphs by Data Mining

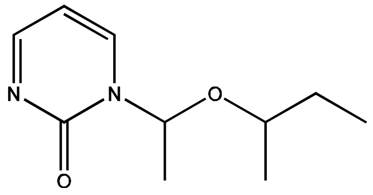
- Our methodology on graph index:
  - Identify frequent structures in the database, the frequent structures are subgraphs that appear quite often in the graph database
  - Prune redundant frequent structures to maintain a small set of discriminative structures
  - Create an inverted index between discriminative frequent structures and graphs in the database

IDEAS: Indexing with Two Constraints

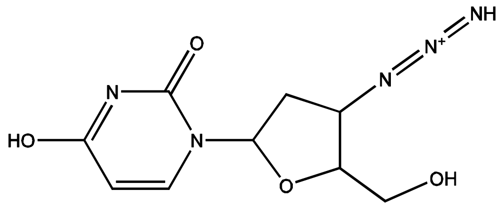


Why Discriminative Subgraphs?

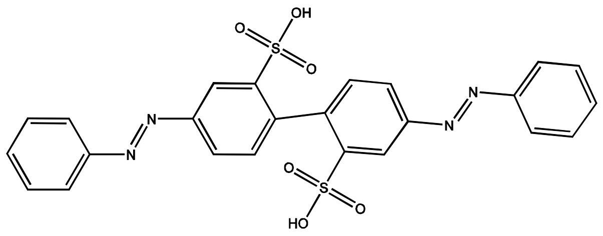
Sample database



(a)



(b)



(c)

- All graphs contain structures: C, C-C, C-C-C
- Why bother indexing these redundant frequent structures?
  - Only index structures that provide more information than existing structures

Discriminative Structures

- Pinpoint the most useful frequent structures
  - Given a set of structures $f_1, f_2, \ldots f_n$ and a new structure x, we measure the extra indexing power provided by x,

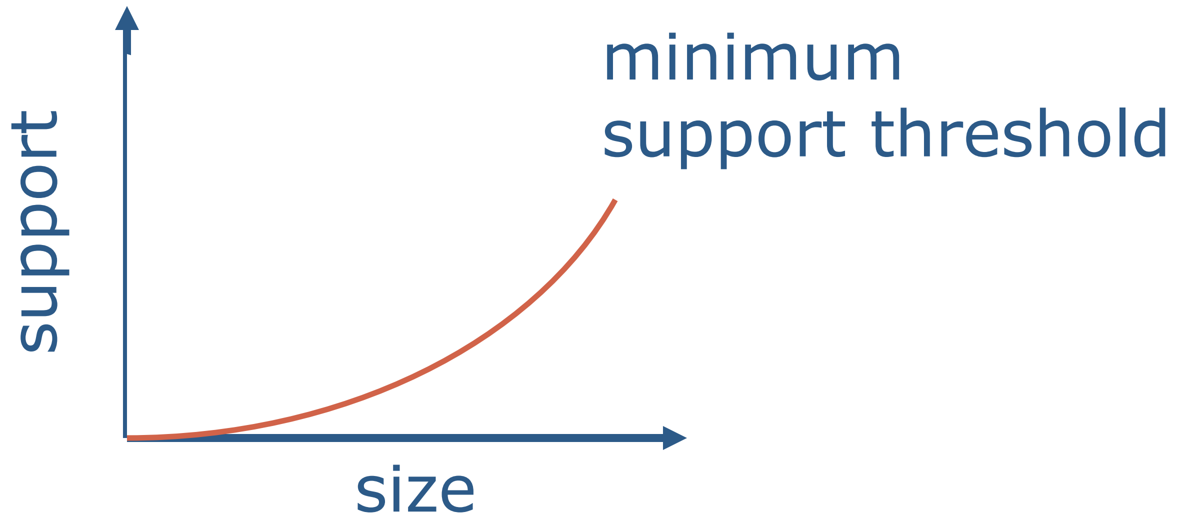
$$P(x \mid f_1, f_2, \ldots f_n), f_i \subset x.$$

When P is small enough, x is a discriminative structure and should be included in the index

- Index discriminative frequent structures only
  - Reduce the index size by an order of magnitude

Why Frequent Structures?

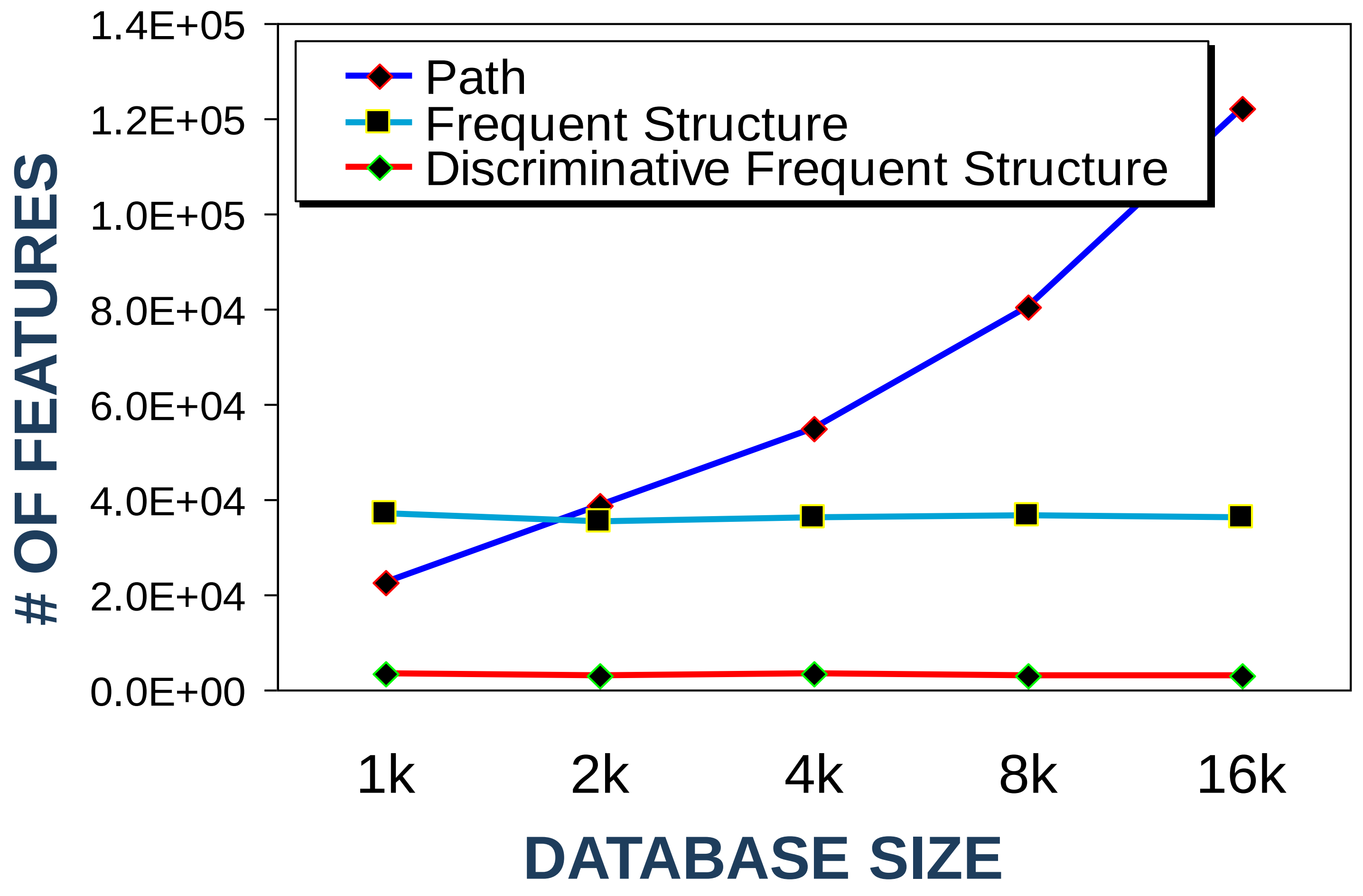
- We cannot index (or even search) all of substructures
- Large structures will likely be indexed well by their substructures
- Size-increasing support threshold



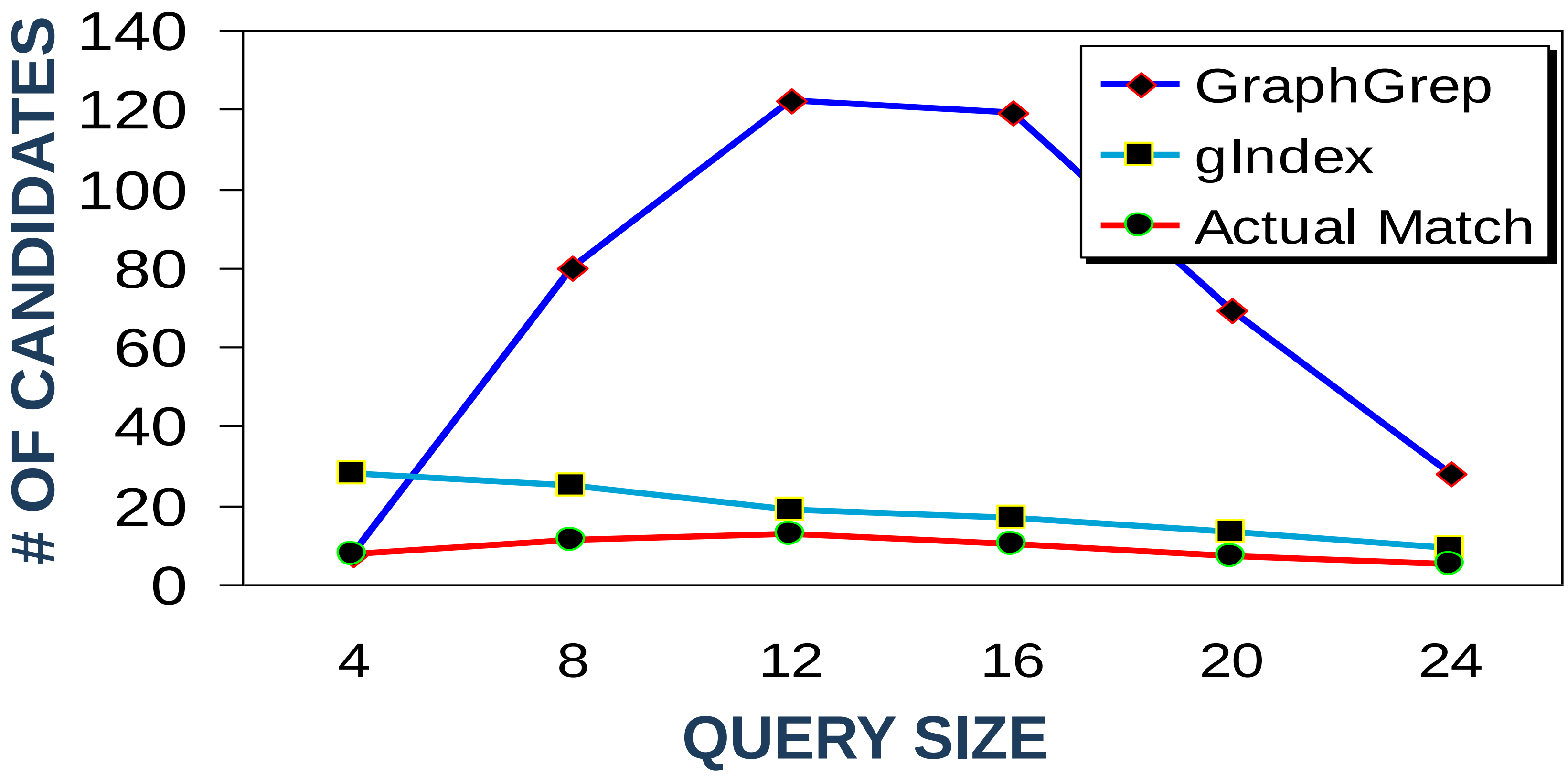
Experimental Setting

- The AIDS antiviral screen compound dataset from NCI/NIH, containing 43,905 chemical compounds
- Query graphs are randomly extracted from the dataset
- GraphGrep: maximum length (edges) of paths is set at 10
- gIndex: maximum size (edges) of structures is set at 10

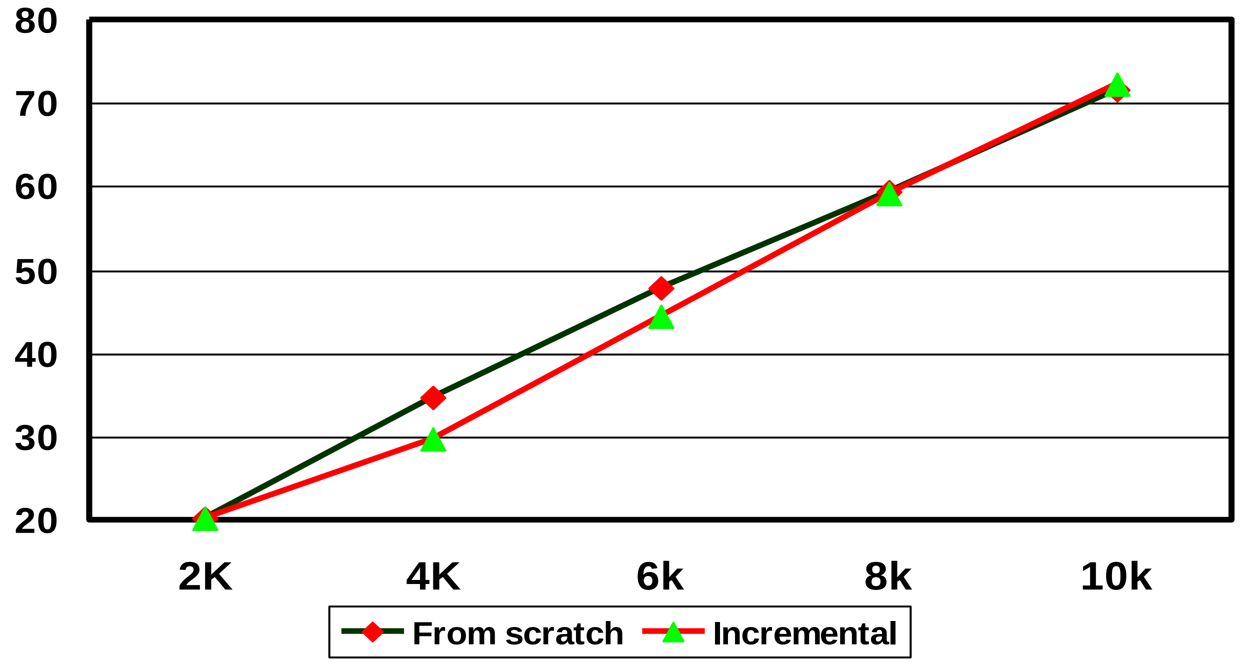
Experiments: Index Size



Experiments: Answer Set Size



Experiments: Incremental Maintenance



Frequent structures are stable to database updating

Index can be built based on a small portion of a graph database, but be used for the whole database

Mining Graph/Network Data: Part I

- Graph / Network Data
- Graph Pattern Mining
- Ranking on Graph / Network
- Summary

Ranking on Graph / Network

- PageRank
- Personalized PageRank

The History of PageRank

- PageRank was developed by Larry Page (hence the name *Page*-Rank) and Sergey Brin.

- It is first as part of a research project about a new kind of search engine. That project started in 1995 and led to a functional prototype in 1998.

- Shortly after, Page and Brin founded Google.

Ranking web pages

- Web pages are not equally "important"
  - www.cnn.com vs. a personal webpage
- Inlinks as votes
  - The more inlinks, the more important
- Are all inlinks equal?
  - Recursive question!

Simple recursive formulation

- Each link's vote is proportional to the importance of its source page
- If page P with importance x has n outlinks, each link gets x/n votes
- Page P's own importance is the sum of the votes on its inlinks

Matrix formulation

- Matrix **M** has one row and one column for each web page
- Suppose page j has n outlinks
  - If j -> i, then $M_{ij}=1/n$
  - Else $M_{ij}=0$
- **M** is a column stochastic matrix
  - Columns sum to 1
- Suppose **r** is a vector with one entry per web page
  - r_i is the importance score of page i
  - Call it the rank vector
  - $|\mathbf{r}| = 1$

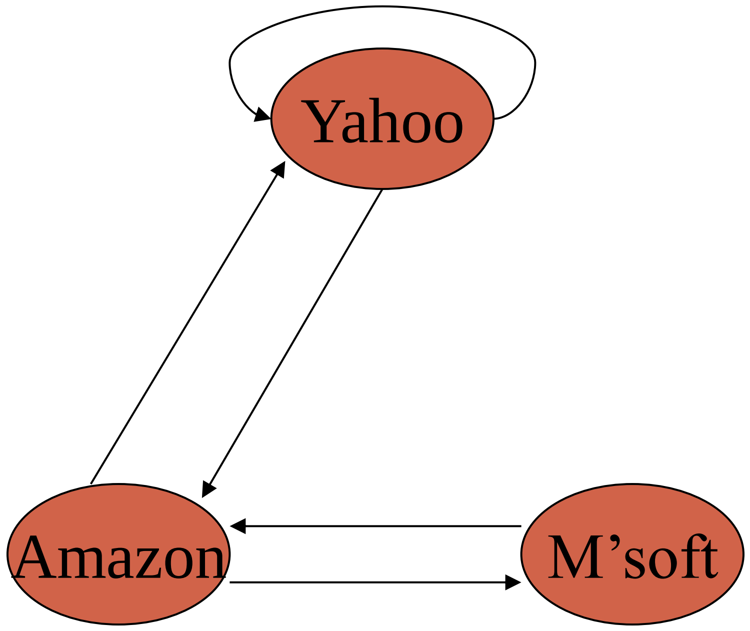
Eigenvector formulation

- The flow equations can be written

$$\mathbf{r} = \mathbf{M}\mathbf{r}$$

- So the rank vector is an eigenvector of the stochastic web matrix
  - In fact, its first or principal eigenvector, with corresponding eigenvalue 1

Example



$$
\begin{array}{c|ccc|}
 & y & a & m \\
\hline
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 1 \\
m & 0 & 1/2 & 0 \\
\hline
\end{array}
$$

r = Mr

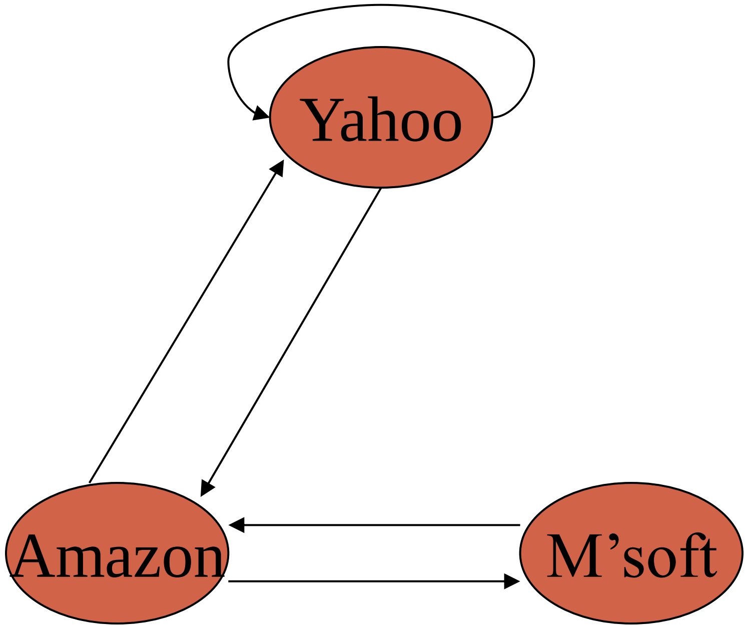
$$y = y/2 + a/2$$
$$a = y/2 + m$$
$$m = a/2$$

$$
\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}
$$

Power Iteration method

- Simple iterative scheme (aka relaxation)
- Suppose there are N web pages
- Initialize: $\mathbf{r}^0 = [1/N,\ldots.,1/N]^T$
- Iterate: $\mathbf{r}^{k+1} = \mathbf{M}\mathbf{r}^k$
- Stop when $|\mathbf{r}^{k+1} - \mathbf{r}^k|_1 < \varepsilon$
  - $|\mathbf{x}|_1 = \sum_{1\le i\le N} |x_i|$ is the L_1 norm
  - Can use any other vector norm e.g., Euclidean

Power Iteration Example



$$\begin{array}{c|ccc|}
 & y & a & m \\
\hline
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 1 \\
m & 0 & 1/2 & 0 \\
\hline
\end{array}$$

$$\begin{array}{cccccccc}
y & & 1/3 & 1/3 & 5/12 & 3/8 & & 2/5 \\
a & = & 1/3 & 1/2 & 1/3 & 11/24 & \ldots & 2/5 \\
m & & 1/3 & 1/6 & 1/4 & 1/6 & & 1/5 \\
 & & \boldsymbol{r}_0 & \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 & \ldots & \boldsymbol{r}^*
\end{array}$$

Random Walk Interpretation

- Imagine a random web surfer
  - At any time t, surfer is on some page P
  - At time t+1, the surfer follows an outlink from P uniformly at random
  - Ends up on some page Q linked from P
  - Process repeats indefinitely
- Let **p**(t) be a vector whose ith component is the probability that the surfer is at page i at time t
  - **p**(t) is a probability distribution on pages

The stationary distribution

- Where is the surfer at time t+1?
  - Follows a link uniformly at random
  - **p**(t+1) = **Mp**(t)
- Suppose the random walk reaches a state such that **p**(t+1) = **Mp**(t) = **p**(t)
  - Then **p**(t) is called a stationary distribution for the random walk
- Our rank vector **r** satisfies **r** = **Mr**
  - So it is a stationary distribution for the random surfer

Existence and Uniqueness

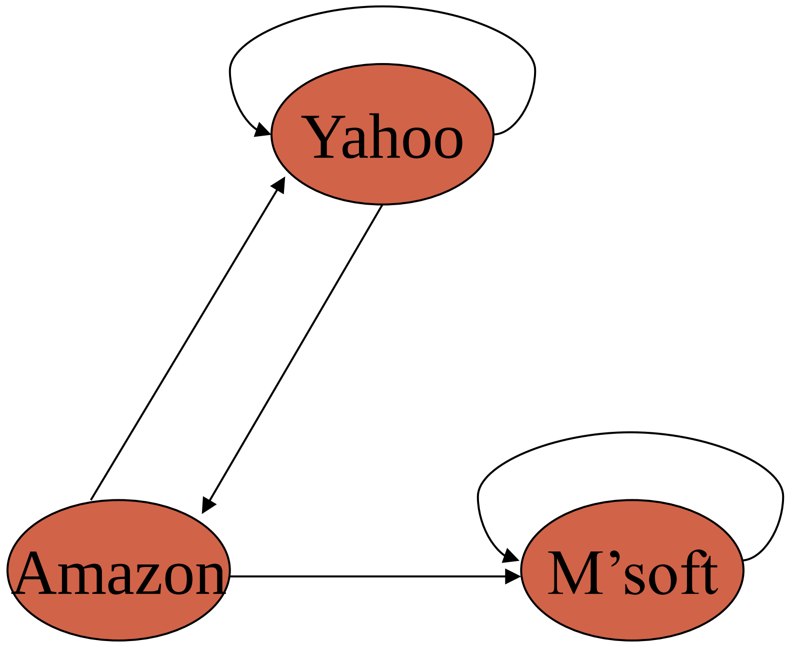
A central result from the theory of random walks (aka Markov processes):

For graphs that satisfy certain conditions, the stationary distribution is unique and eventually will be reached no matter what the initial probability distribution at time t = 0.

Spider traps

- A group of pages is a spider trap if there are no links from within the group to outside the group
  - Random surfer gets trapped
- Spider traps violate the conditions needed for the random walk theorem

Microsoft becomes a spider trap



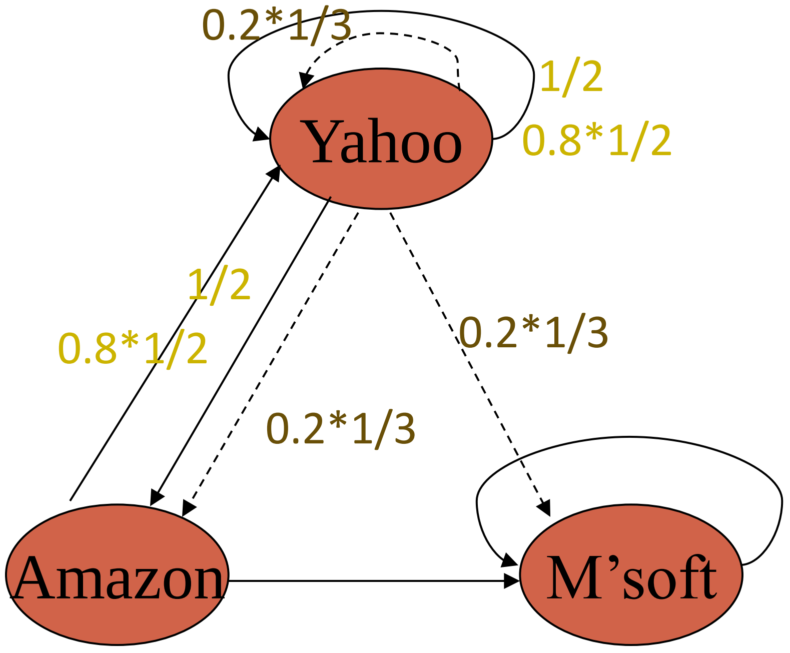
$$\begin{array}{c|ccc|}
 & y & a & m \\
\hline
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 0 \\
m & 0 & 1/2 & 1 \\
\hline
\end{array}$$

$$\begin{array}{ccccccccc}
y & & 1/3 & 1/3 & 1/4 & 5/24 & & 0 \\
a & = & 1/3 & 1/6 & 1/6 & 1/8 & \ldots & 0 \\
m & & 1/3 & 1/2 & 7/12 & 2/3 & & 1
\end{array}$$

Random teleports

- The Google solution for spider traps
- At each time step, the random surfer has two options:
  - With probability β, follow a link at random
  - With probability 1-β, jump to some page uniformly at random
  - Common values for β are in the range 0.8 to 0.9
- Surfer will teleport out of spider trap within a few time steps

Random teleports ($\beta = 0.8$)

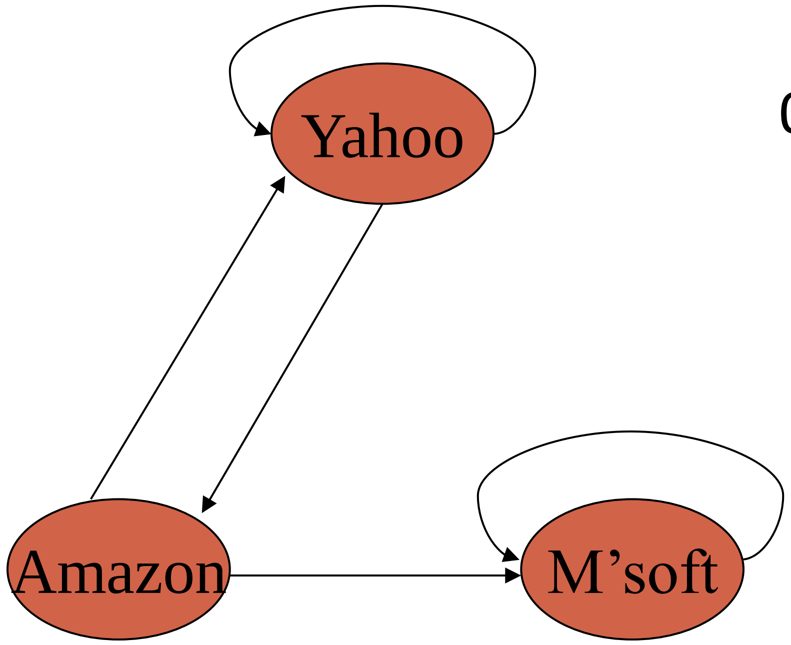


$$\begin{array}{c|c|} & y \\ y & 1/2 \\ a & 1/2 \\ m & 0 \end{array} \quad 0.8* \begin{array}{|c|} y \\ 1/2 \\ 1/2 \\ 0 \end{array} + 0.2* \begin{array}{|c|} y \\ 1/3 \\ 1/3 \\ 1/3 \end{array}$$

$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$\begin{array}{c|ccc|} y & 7/15 & 7/15 & 1/15 \\ a & 7/15 & 1/15 & 1/15 \\ m & 1/15 & 7/15 & 13/15 \end{array}$$

Random teleports ($\beta = 0.8$)



$$0.8 \begin{array}{|ccc|} \hline 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \\ \hline \end{array} + 0.2 \begin{array}{|ccc|} \hline 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ \hline \end{array}$$

$$\begin{array}{c|ccc|} \hline y & 7/15 & 7/15 & 1/15 \\ a & 7/15 & 1/15 & 1/15 \\ m & 1/15 & 7/15 & 13/15 \\ \hline \end{array}$$

$$\begin{matrix} y \\ a \\ m \end{matrix} = \begin{bmatrix} 0.333 \\ 0.333 \\ 0.333 \end{bmatrix} \begin{bmatrix} 0.333 \\ 0.200 \\ 0.467 \end{bmatrix} \begin{bmatrix} 0.280 \\ 0.200 \\ 0.520 \end{bmatrix} \begin{bmatrix} 0.259 \\ 0.179 \\ 0.563 \end{bmatrix} \ldots \begin{bmatrix} 7/33 \\ 5/33 \\ 21/33 \end{bmatrix}$$

Matrix formulation

- Suppose there are N pages
  - Consider a page j, with set of outlinks O(j)
  - We have $M_{ij} = 1/|O(j)|$ when j->i and $M_{ij} = 0$ otherwise
  - The random teleport is equivalent to
    - adding a teleport link from j to every other page with probability $(1-\beta)/N$
    - reducing the probability of following each outlink from $1/|O(j)|$ to $\beta/|O(j)|$
    - Equivalent: tax each page a fraction $(1-\beta)$ of its score and redistribute evenly

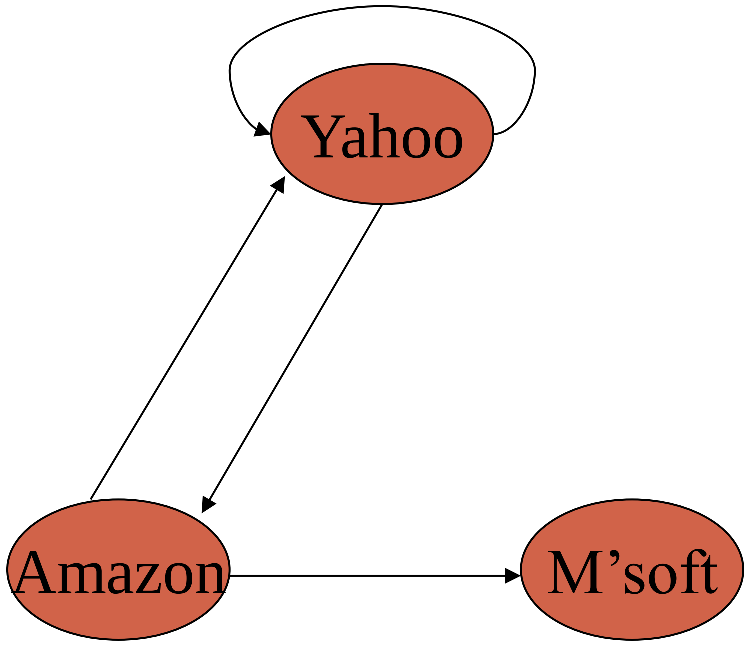
PageRank

- Construct the N-by-N matrix **A** as follows
  - $A_{ij} = \beta M_{ij} + (1-\beta)/N$
- Verify that **A** is a stochastic matrix
- The page rank vector **r** is the principal eigenvector of this matrix
  - satisfying $\mathbf{r} = \mathbf{A}\mathbf{r}$
- Equivalently, **r** is the stationary distribution of the random walk with teleports

Dead ends

- Pages with no outlinks are "dead ends" for the random surfer
  - Nowhere to go on next step

Microsoft becomes a dead end



$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$\begin{array}{c|ccc|} \text{y} & 7/15 & 7/15 & 1/15 \\ \text{a} & 7/15 & 1/15 & 1/15 \\ \text{m} & 1/15 & 7/15 & 1/15 \end{array}$$

Non-stochastic!

$$\begin{array}{l} \text{y} \\ \text{a} \\ \text{m} \end{array} = \begin{array}{c} 1/3 \\ 1/3 \\ 1/3 \end{array} \quad \begin{array}{c} 1/3 \\ 0.2 \\ 0.2 \end{array} \quad \ldots \quad \begin{array}{c} 0 \\ 0 \\ 0 \end{array}$$

Dealing with dead-ends

- Teleport
  - Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly
- Prune and propagate
  - Preprocess the graph to eliminate dead-ends
  - Might require multiple passes
  - Compute page rank on reduced graph
  - Approximate values for deadends by propagating values from reduced graph

Computing PageRank

- Key step is matrix-vector multiplication
  - $\mathbf{r}^{\text{new}} = \mathbf{A}\mathbf{r}^{\text{old}}$
- Easy if we have enough main memory to hold **A**, $\mathbf{r}^{\text{old}}$, $\mathbf{r}^{\text{new}}$
- Say N = 1 billion pages
  - We need 4 bytes for each entry (say)
  - 2 billion entries for vectors, approx 8GB
  - Matrix A has N^2 entries
    - 10^{18} is a large number!

Rearranging the equation

$\mathbf{r} = \mathbf{A}\mathbf{r}$, where

$A_{ij} = \beta M_{ij} + (1-\beta)/N$

$r_i = \sum_{1 \le j \le N} A_{ij}\, r_j$

$r_i = \sum_{1 \le j \le N} [\beta M_{ij} + (1-\beta)/N]\, r_j$

$= \beta \sum_{1 \le j \le N} M_{ij}\, r_j + (1-\beta)/N \sum_{1 \le j \le N} r_j$

$= \beta \sum_{1 \le j \le N} M_{ij}\, r_j + (1-\beta)/N$, since $|\mathbf{r}| = 1$

$\mathbf{r} = \beta \mathbf{M}\mathbf{r} + [(1-\beta)/N]_N$

where $[x]_N$ is an N-vector with all entries x

Sparse matrix formulation

- We can rearrange the page rank equation:
  - $\mathbf{r} = \beta\mathbf{M}\mathbf{r} + [(1-\beta)/N]_N$
  - $[(1-\beta)/N]_N$ is an N-vector with all entries $(1-\beta)/N$
- **M** is a sparse matrix!
  - 10 links per node, approx 10N entries
- So in each iteration, we need to:
  - Compute $\mathbf{r}^{new} = \beta\mathbf{M}\mathbf{r}^{old}$
  - Add a constant value $(1-\beta)/N$ to each entry in $\mathbf{r}^{new}$

Sparse matrix encoding

- Encode sparse matrix using only nonzero entries
  - Space proportional roughly to number of links
  - say 10N, or 4*10*1 billion = 40GB
  - still won't fit in memory, but will fit on disk

source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

Basic Algorithm

- Assume we have enough RAM to fit $\mathbf{r}^{new}$, plus some working memory
  - Store $\mathbf{r}^{old}$ and matrix $\mathbf{M}$ on disk

Basic Algorithm:

- Initialize: $\mathbf{r}^{old} = [1/N]_N$
- Iterate:
  - Update: Perform a sequential scan of $\mathbf{M}$ and $\mathbf{r}^{old}$ to update $\mathbf{r}^{new}$
  - Write out $\mathbf{r}^{new}$ to disk as $\mathbf{r}^{old}$ for next iteration
  - Every few iterations, compute $|\mathbf{r}^{new}-\mathbf{r}^{old}|$ and stop if it is below threshold
    - Need to read in both vectors into memory

Personalized PageRank

- Query-dependent Ranking
  - For a query webpage q, which webpages are most important to q?
  - The relative important webpages to different queries would be different

Calculation of P-PageRank

- Recall PageRank calculation:
  - $\mathbf{r} = \beta\mathbf{Mr} + [(1-\beta)/N]_N$ or
  - $\mathbf{r} = \beta\mathbf{Mr} + (1-\beta)\, r_0$, where $r_0 = \begin{pmatrix} 1/N \\ 1/N \\ \dots \\ 1/N \end{pmatrix}$
- For P-PageRank
  - Replace r_0 with $r_0 = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{pmatrix} \leftarrow \boxed{\text{qth webpage}}$

Mining Graph/Network Data: Part I

- Graph / Network Data
- Graph Pattern Mining
- Ranking on Graph / Network
- Summary

Summary

- Graph / Network Data
  - Adjacency matrix
- Graph Pattern Mining
  - Frequent subgraph mining
    - gSpan
  - Graph search
    - gindex
- Ranking on Graph / Network
  - PageRank
  - Personalized PageRank