Mining, Indexing, and Similarity Search in Graphs and Complex Structures

Jiawei Han **Xifeng Yan**
Department of Computer Science
University of Illinois at Urbana-Champaign
Philip S. Yu
IBM T. J. Watson Research Center

Outline

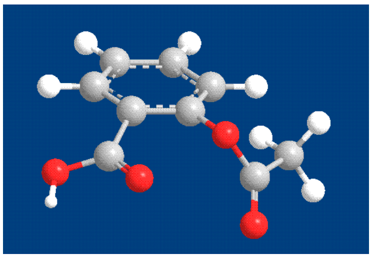
- Scalable pattern mining in graph data sets
  - Frequent subgraph pattern mining
  - Constraint-based graph pattern mining
  - Graph clustering, classification, and compression
- Searching graph databases
  - Graph indexing methods
  - Similarity search in graph databases
- Application and exploration with graph mining
  - Biological and social network analysis
  - Mining software systems: bug isolation & performance tuning
- Conclusions and future work

Why Graph Mining and Searching?

- 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

---

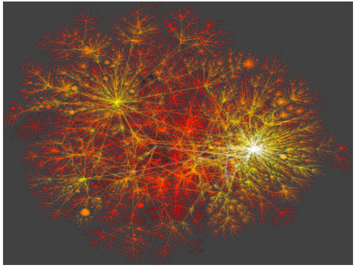
Graph, Graph, Everywhere



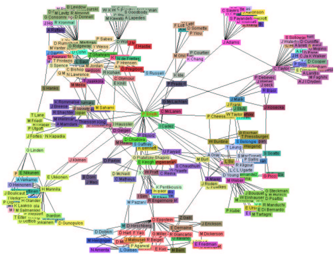
Aspirin

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

Yeast protein interaction network



Internet



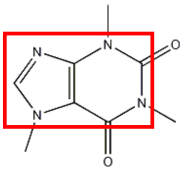
Co-author network

Graph Pattern Mining

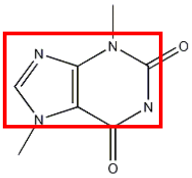
- *Frequent* subgraphs
  - A (sub)graph is ***frequent*** if its *support* (occurrence frequency) in a given dataset is no less than a *minimum support* threshold
- Applications of graph pattern mining
  - Mining biochemical structures
  - Program control flow analysis
  - Mining XML structures or Web communities
  - Building blocks for graph classification, clustering, compression, comparison, and correlation analysis

Example: Frequent Subgraphs

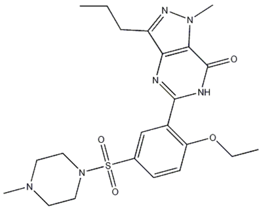
CHEMICAL COMPOUNDS



(a) caffeine

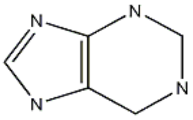


(b) diurobromine



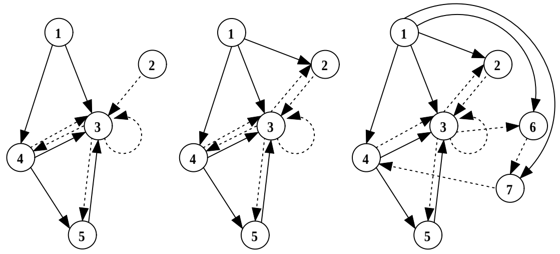
(c) viagra …

FREQUENT SUBGRAPH



Example (II)

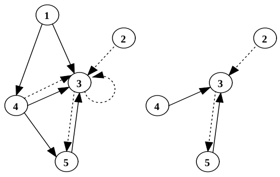
GRAPH DATASET



1: makepat
2: esc
3: addstr
4: getccl
5: dodash
6: in_set_2
7: stclose

(1) (2) (3)

FREQUENT PATTERNS (MIN SUPPORT IS 2)



(1) (2)

Graph Mining Algorithms

- Incomplete beam search – Greedy (Subdue)
- Inductive logic programming (WARMR)
- Graph theory based approaches
  - Apriori-based approach
  - Pattern-growth approach

SUBDUE (Holder et al. KDD’94)

- Start with single vertices
- Expand best substructures with a new edge
- Limit the number of best substructures
  - Substructures are evaluated based on their ability to compress input graphs
  - Using minimum description length (DL)
  - Best substructure *S* in graph *G* minimizes: DL(S) + DL(G\S)
- Terminate until no new substructure is discovered

WARMR (Dehaspe et al. KDD’98)

- Graphs are represented by Datalog facts
  - *atomel(C, A1, c), bond (C, A1, A2, BT), atomel(C, A2, c) : a carbon atom bound to a carbon atom with bond type BT*
- WARMR: the first general purpose ILP system
- Level-wise search
- Simulate Apriori for frequent pattern discovery

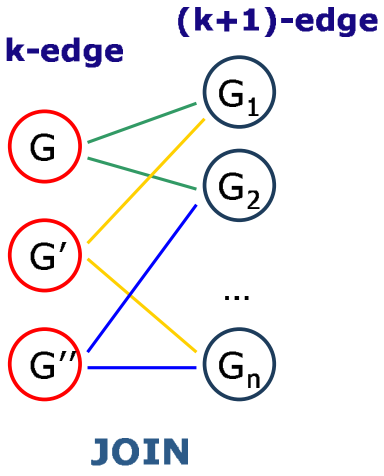
Frequent Subgraph Mining Approaches

- 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)

Properties of Graph Mining Algorithms

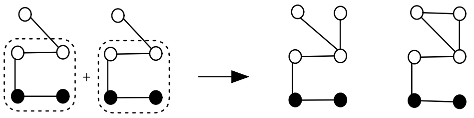
- Search order
  - breadth vs. depth
- Generation of candidate subgraphs
  - apriori vs. pattern growth
- Elimination of duplicate subgraphs
  - passive vs. active
- Support calculation
  - embedding store or not
- Discover order of patterns
  - path → tree → graph

Apriori-Based Approach

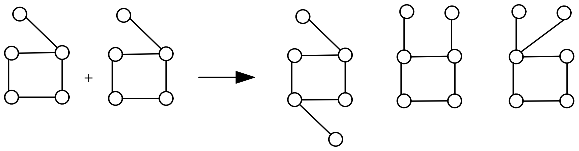


Apriori-Based, Breadth-First Search

- Methodology: breadth-search, joining two graphs



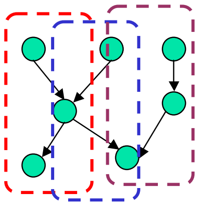
- AGM (Inokuchi, et al. PKDD’00)
  - generates new graphs with one more node



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

PATH (Vanetik and Gudes ICDM'02, '04)

- Apriori-based approach
- Building blocks: edge-disjoint path



A graph with 3 edge-disjoint paths

- construct frequent paths
- construct frequent graphs with 2 edge-disjoint paths
- construct graphs with k+1 edge-disjoint paths from graphs with k edge-disjoint paths
- repeat

FFSM (Huan, et al. ICDM'03)

X			
2	Y		
	1	X	
	1		Z

+

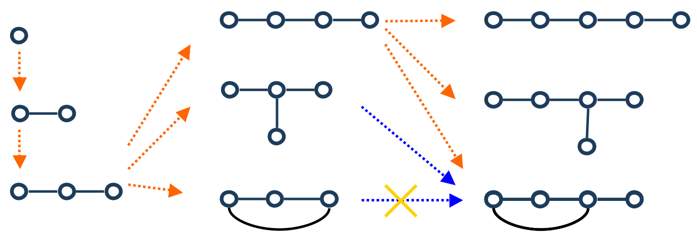
X			
2	Y		
	1	X	
		1	Z

→

X			
2	Y		
	1	X	
	1	1	Z

- Represent graphs using canonical adjacency matrix (CAM)
- Join two CAMs or extend a CAM to generate a new graph
- Store the embeddings of CAMs
  - All of the embeddings of a pattern in the database
  - Can derive the embeddings of newly generated CAMs

Pattern Growth Method



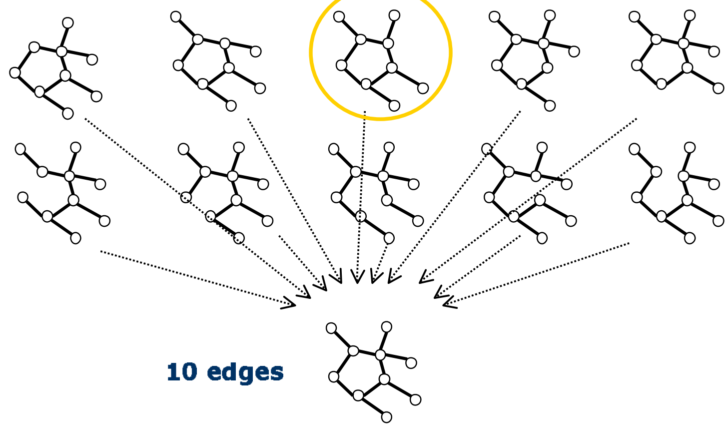
- detect duplicates
- avoid duplicates

MoFa (Borgelt and Berthold ICDM’02)

- Extend graphs by adding a new edge
- Store embeddings of discovered frequent graphs
  - Fast support calculation
  - Also used in other later developed algorithms such as FFSM and GASTON
  - Expensive Memory usage
- Local structural pruning

Duplicate Graphs

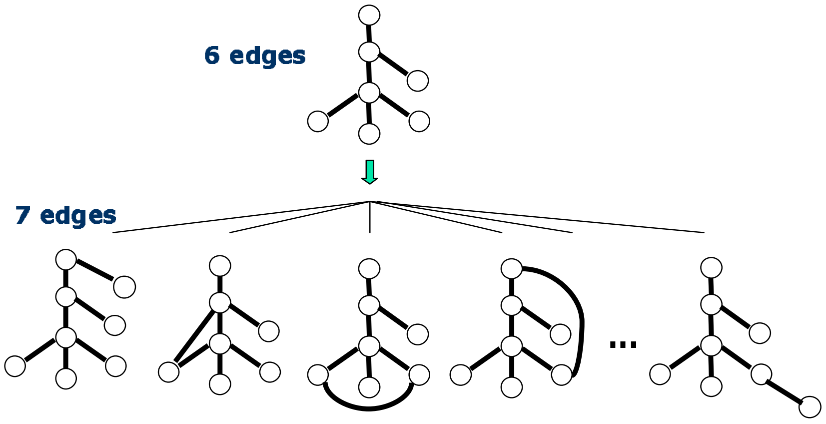
9 edges



10 edges

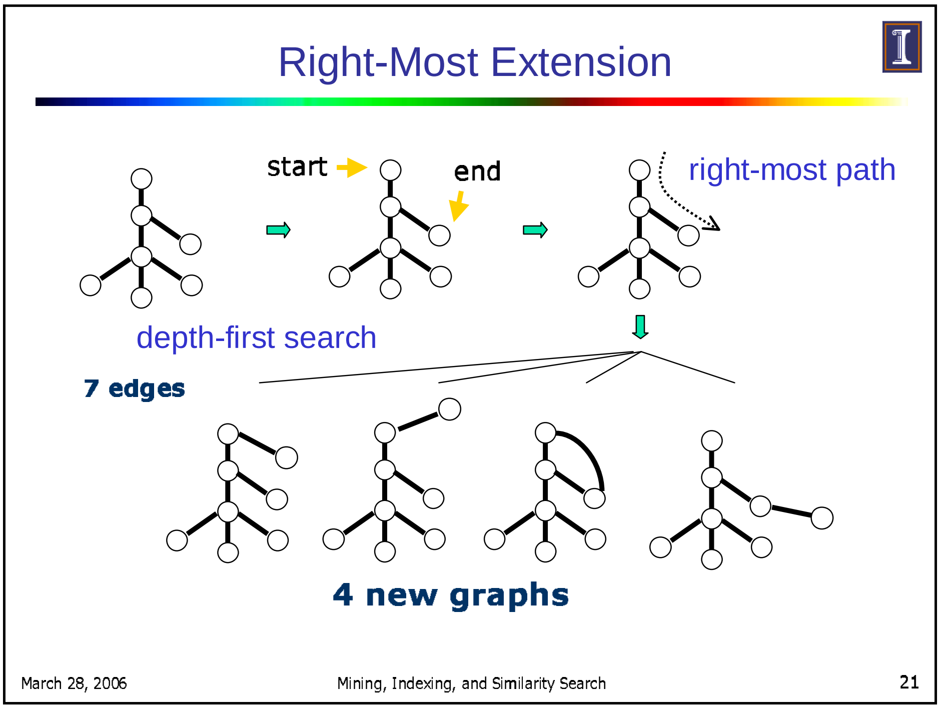
Free Extension

6 edges



7 edges

22 new graphs



Right-Most Extension

start

end

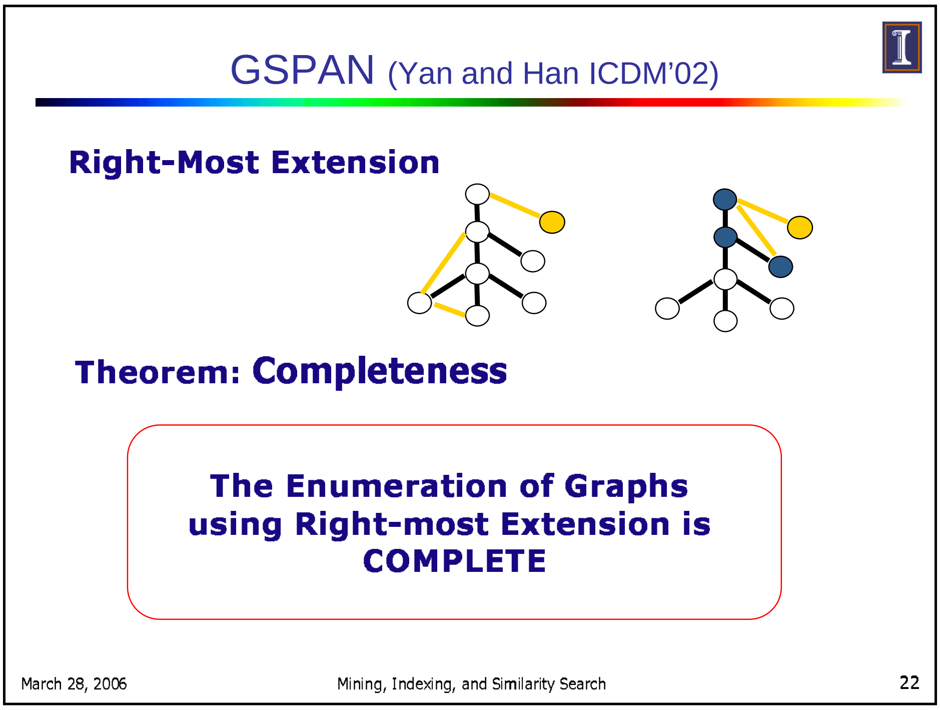
right-most path

depth-first search

7 edges

4 new graphs

March 28, 2006 Mining, Indexing, and Similarity Search 21



GSPAN (Yan and Han ICDM'02)

Right-Most Extension

Theorem: Completeness

The Enumeration of Graphs using Right-most Extension is COMPLETE

March 28, 2006 Mining, Indexing, and Similarity Search 22

Graph Sequentialization

Canonical labeling system: DFS coding

$$G \rightarrow S$$

graph edge sequence

Goals: 1. any prefix of a canonical label is canonical
2. follow right most extension

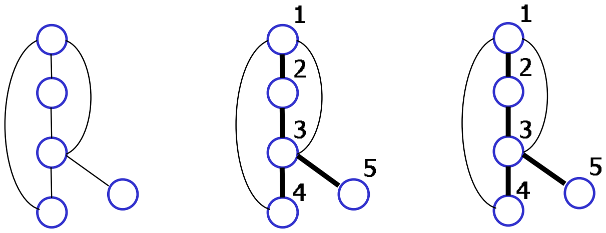
$$\forall s \in S,\ \exists\, t \in S,\ \ t + elem = s$$

$$\Leftrightarrow \forall s \in S,\ \forall t,\ \ t\ is\ a\ prefix\ of\ s, t \in S$$

$$\Leftrightarrow \forall s \notin S,\ \forall t,\ \ s + t \notin S$$

DFS Coding & Labelling

DFS coding: flatten a graph into a sequence based on depth-first search



depth first search

(1,2) (2,3) (3,1) (3,4) (4,1) (3,5)

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
The DFS code of a graph extended from a Non-minimum DFS code is NOT MINIMUM

GASTON (Nijssen and Kok KDD’04)

- Extend graphs directly
- Store embeddings
- Separate the discovery of different types of graphs
    - path → tree → graph
    - Simple structures are easier to mine and duplication detection is much simpler

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 **423** chemical compounds which are confirmed to be active in an AIDS antiviral screen dataset, there are around **1,000,000** frequent graph patterns if the minimum support is 5%

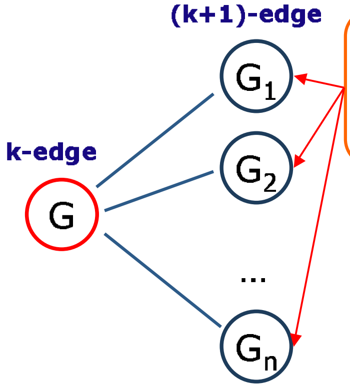
Closed Frequent Graphs

- Motivation: Handling graph pattern explosion problem
- Closed frequent graph
  - A frequent graph G is *closed* if there exists no supergraph of G that carries the same support as G
- If some of G’s subgraphs have the same support, it is unnecessary to output these subgraphs (nonclosed graphs)
- *Lossless compression:* still ensures that the mining result is complete

---

CLOSEGRAPH (Yan & Han, KDD’03)

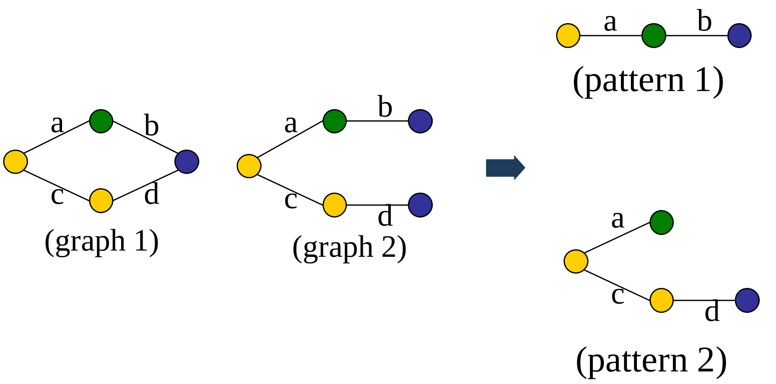
A Pattern-Growth Approach



At what condition, can we stop searching their children i.e., early termination?

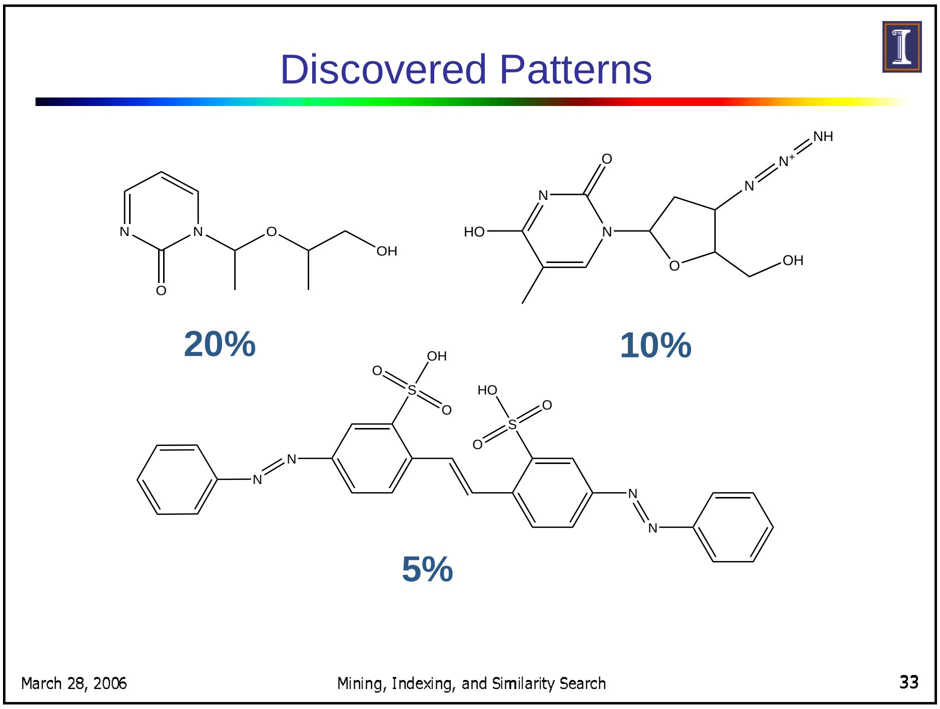
If G and G′ are frequent, G is a subgraph of G′. If **in any part of the graph in the dataset where G occurs, G′ also occurs**, then we need not grow G, since none of G’s children will be closed except those of G′.

Handling Tricky Exception Cases



Experimental Result

- The AIDS antiviral screen compound dataset from NCI/NIH
- The dataset contains 43,905 chemical compounds
- Among these 43,905 compounds, **423 of them belong to CA, 1081 are of CM**, and the remainings are in class CI

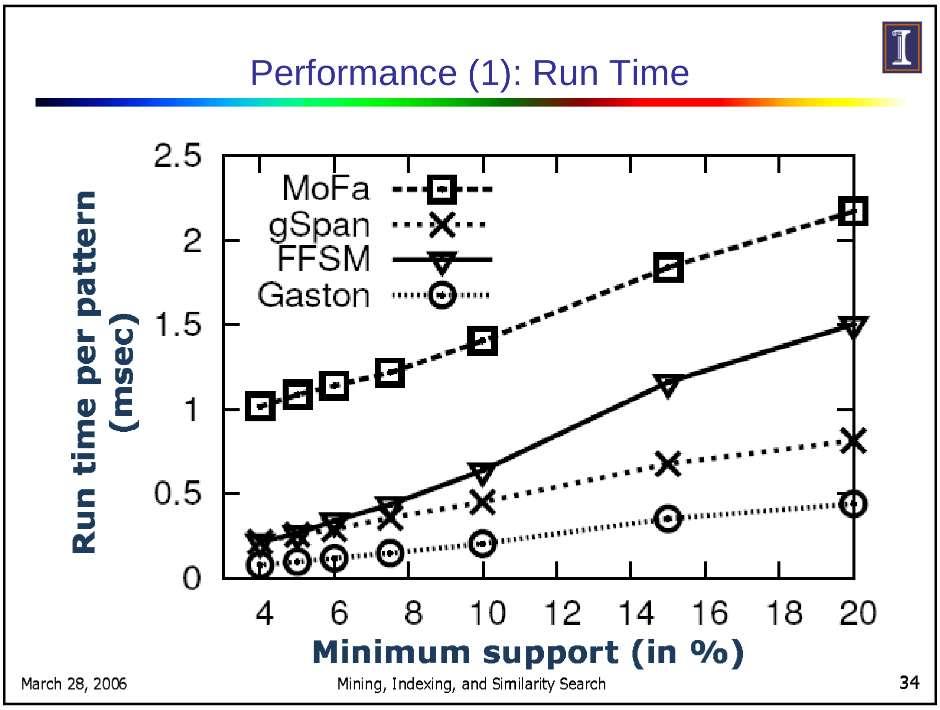


Discovered Patterns

20%

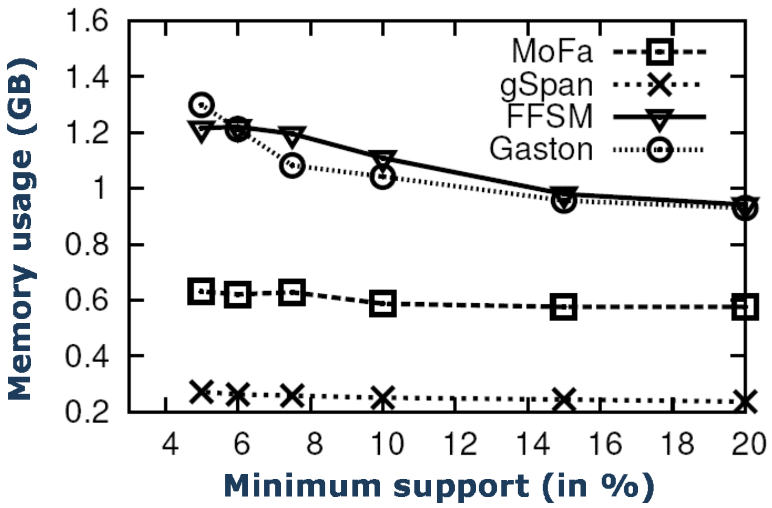
10%

5%

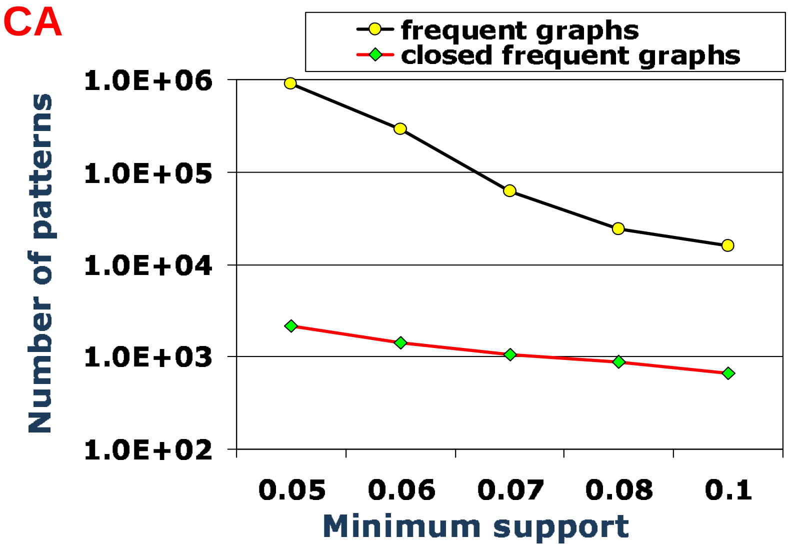


Performance (1): Run Time

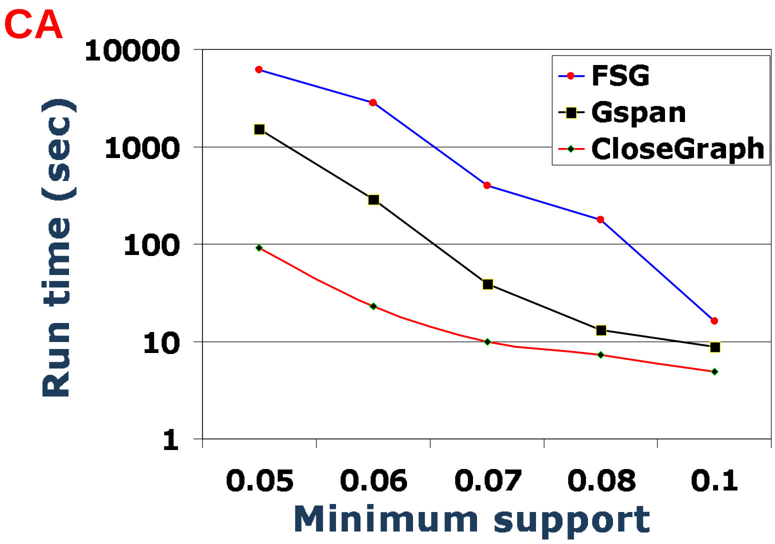
Performance (2): Memory Usage



Number of Patterns: Frequent vs. Closed



Runtime: Frequent vs. Closed



Outline

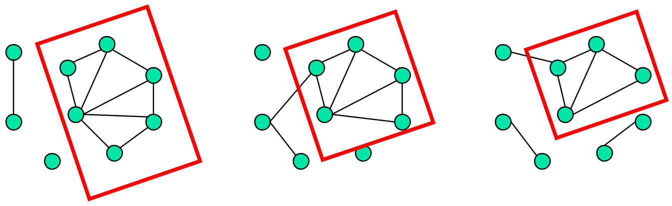
- Scalable pattern mining in graph data sets
  - Frequent subgraph pattern mining
  - Constraint-based graph pattern mining
  - Graph clustering, classification, and compression
- Searching graph databases
  - Graph indexing methods
  - Similarity search in graph databases
- Application and exploration with graph mining
  - Biological and social network analysis
  - Mining computer systems: bug isolation & performance tuning
- Conclusions and future work

Constrained Patterns

- Density
- Diameter
- Connectivity
- Degree
- Min, Max, Avg

Constraint-Based Graph Pattern Mining

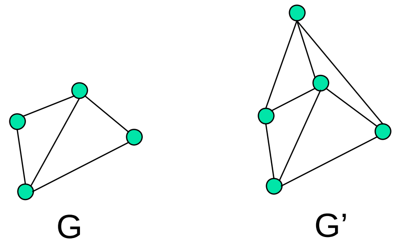
- Highly connected subgraphs in a large graph usually are not artifacts (group, functionality)



- Recurrent patterns discovered in multiple graphs are more robust than the patterns mined from a single graph

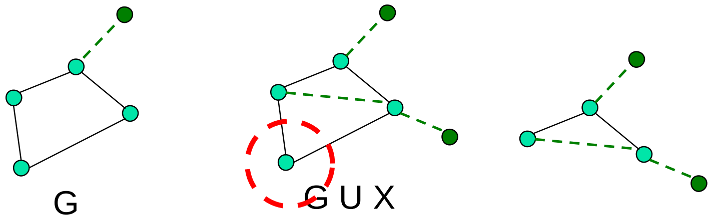
No Downward Closure Property

Given two graphs G and G’, if G is a subgraph of G’, it does not imply that the connectivity of G is less than that of G’, and vice versa.



Minimum Degree Constraint

Let G be a frequent graph and X be the set of edges which can be added to G such that G U e (e ε X) is connected and frequent. Graph G U X is the maximal graph that can be Extended (one step) from the vertices belong to G

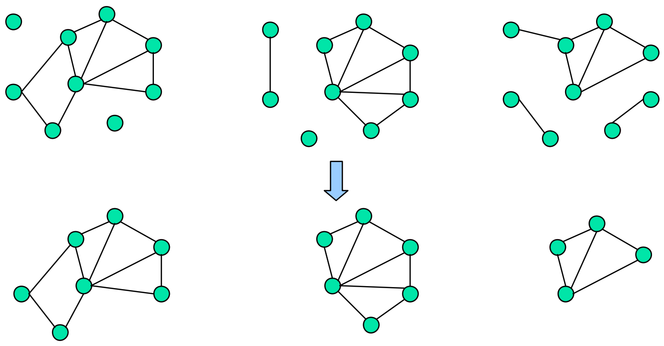


Pattern-Growth Approach

- Find a small frequent candidate graph
  - Remove vertices (shadow graph) whose degree is less than the connectivity
  - Decompose it to extract the subgraphs satisfying the connectivity constraint
  - Stop decomposing when the subgraph has been checked before
- Extend this candidate graph by adding new vertices and edges
- Repeat

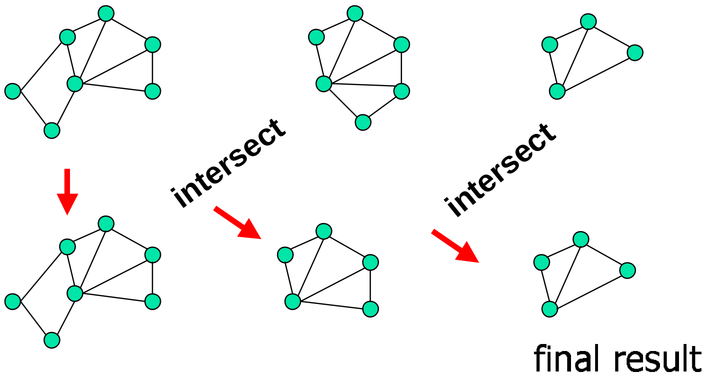
Pattern-Reduction Approach

- Decompose the relational graphs according to the connectivity constraint



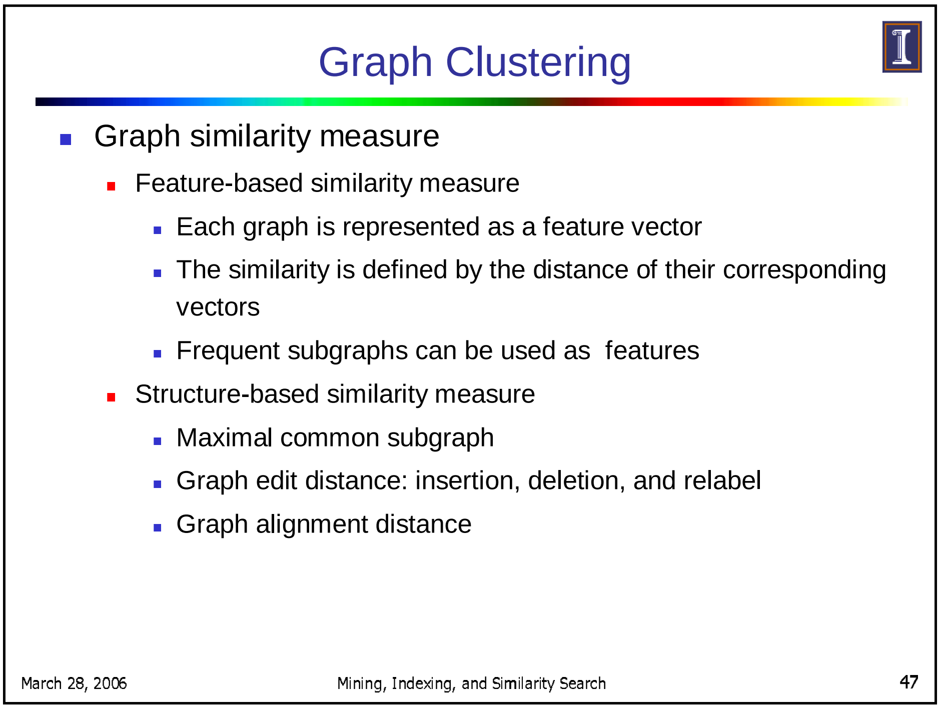
Pattern-Reduction Approach (cont.)

- Intersect them and decompose the resulting subgraphs



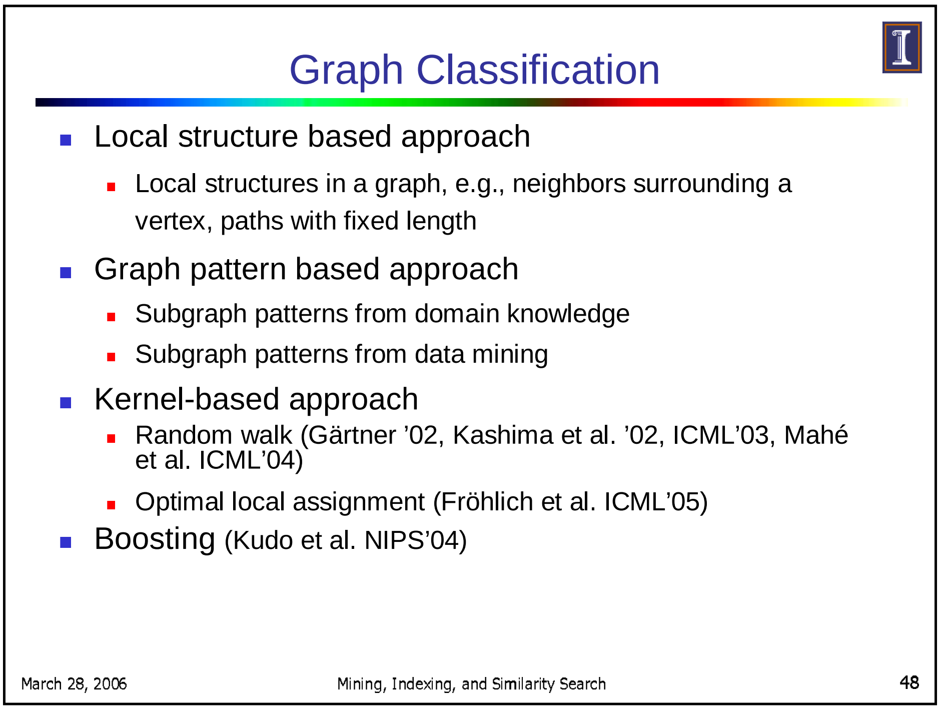
Outline

- Scalable pattern mining in graph data sets
  - Frequent subgraph pattern mining
  - Constraint-based graph pattern mining
  - Graph clustering, classification, and compression
- Searching graph databases
  - Graph indexing methods
  - Similarity search in graph databases
- Application and exploration with graph mining
  - Biological and social network analysis
  - Mining computer systems: bug isolation & performance tuning
- Conclusions and future work



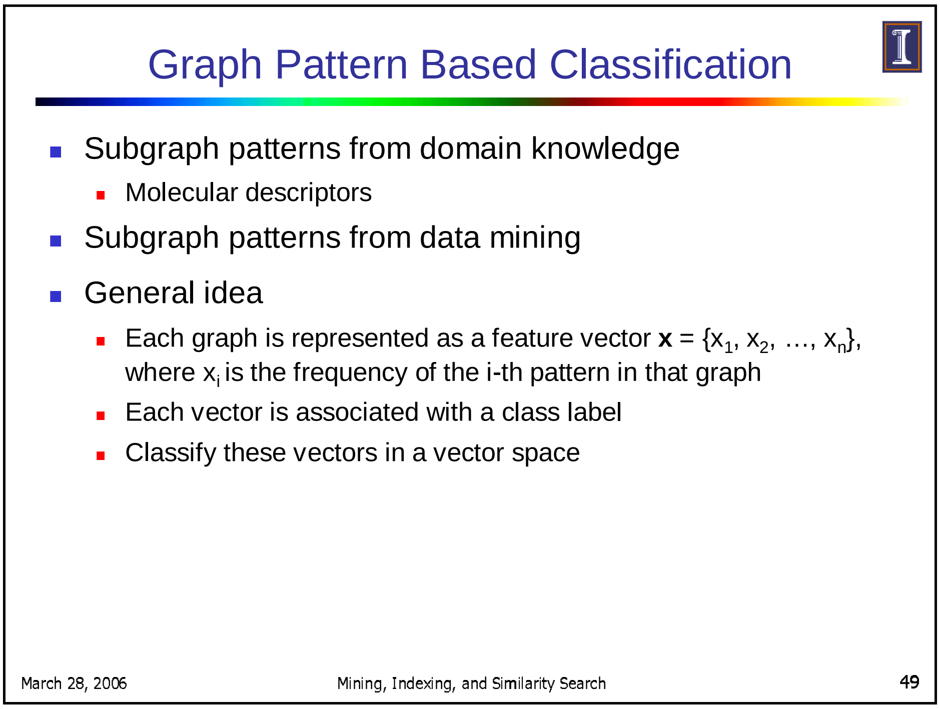
Graph Clustering

- Graph similarity measure
  - Feature-based similarity measure
    - Each graph is represented as a feature vector
    - The similarity is defined by the distance of their corresponding vectors
    - Frequent subgraphs can be used as features
  - Structure-based similarity measure
    - Maximal common subgraph
    - Graph edit distance: insertion, deletion, and relabel
    - Graph alignment distance



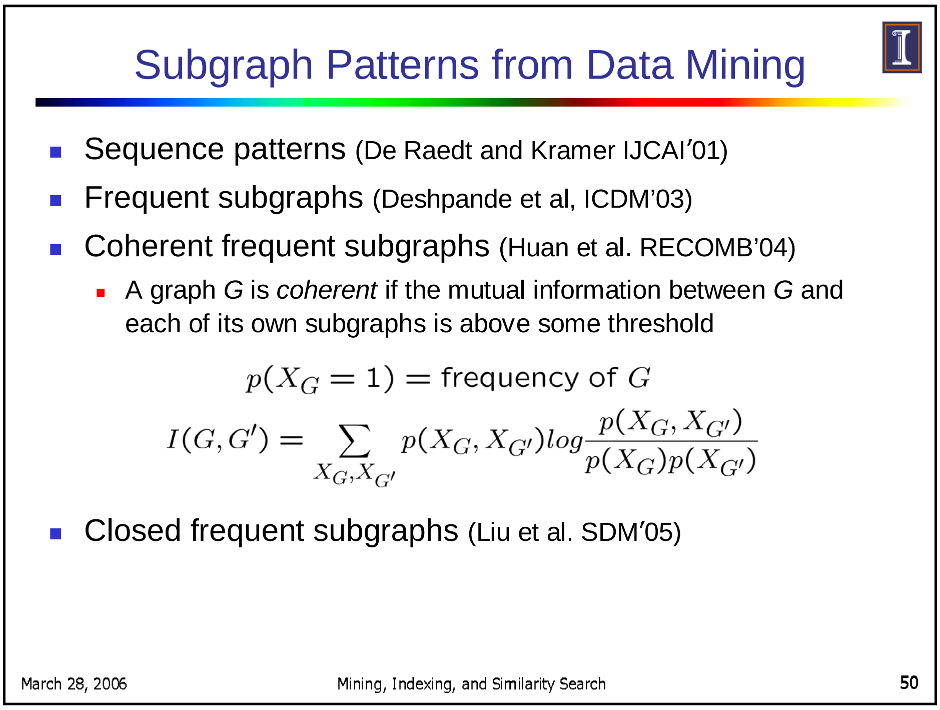
Graph Classification

- Local structure based approach
  - Local structures in a graph, e.g., neighbors surrounding a vertex, paths with fixed length
- Graph pattern based approach
  - Subgraph patterns from domain knowledge
  - Subgraph patterns from data mining
- Kernel-based approach
  - Random walk (Gärtner ’02, Kashima et al. ’02, ICML’03, Mahé et al. ICML’04)
  - Optimal local assignment (Fröhlich et al. ICML’05)
- Boosting (Kudo et al. NIPS’04)



Graph Pattern Based Classification

- Subgraph patterns from domain knowledge
  - Molecular descriptors
- Subgraph patterns from data mining
- General idea
  - Each graph is represented as a feature vector **x** = $\{x_1, x_2, \ldots, x_n\}$, where x_i is the frequency of the i-th pattern in that graph
  - Each vector is associated with a class label
  - Classify these vectors in a vector space



Subgraph Patterns from Data Mining

- Sequence patterns (De Raedt and Kramer IJCAI'01)
- Frequent subgraphs (Deshpande et al, ICDM'03)
- Coherent frequent subgraphs (Huan et al. RECOMB'04)
  - A graph *G* is *coherent* if the mutual information between *G* and each of its own subgraphs is above some threshold

$$p(X_G = 1) = \text{frequency of } G$$

$$I(G, G') = \sum_{X_G, X_{G'}} p(X_G, X_{G'}) log \frac{p(X_G, X_{G'})}{p(X_G)p(X_{G'})}$$

- Closed frequent subgraphs (Liu et al. SDM'05)

Kernel-based Classification

- Random walk
  - Marginalized Kernels (Gärtner ’02, Kashima et al. ’02, ICML’03, Mahé et al. ICML’04)

$$K(G_1, G_2) = \sum_{h_1} \sum_{h_2} p(h_1) p(h_2) K_L(l(h_1), l(h_2))$$

- h_1 and h_2 are paths in graphs G_1 and G_2
- $p(h_1)$ and $p(h_2)$ are probability distributions on paths
- $K_L(l(h_1), l(h_2))$ is a kernel between paths, e.g.,

$$K_L(l_1, l_2) = \begin{cases} 1 & \text{if } l_1 = l_2, \\ 0 & otherwise. \end{cases}$$

Kernel-based Classification

- Optimal local assignment (Fröhlich et al. ICML’05)

$$K(G, G') = \begin{cases} max_{\pi} \sum_{i=1}^{|V(G)|} \boxed{k(v_i, v'_{\pi_i})} & \text{if } |V(G)| \geq |V(G')|, \\ max_{\pi} \sum_{i=1}^{|V(G')|} k(v_{\pi_i}, v'_i) & otherwise. \end{cases}$$

Can be extended to include neighborhood information e.g.

$$k_{nei}(v, v') = k_{atom}(v, v') + \sum_{l=0}^{L} \lambda_l R_l(v, v')$$

where R_l could be an RBF-kernel to measure the similarity of neighborhoods of vertices v and v', λ_l is a damping parameter.

Boosting in Graph Classification

- Decision stumps
  - Simple classifiers in which the final decision is made by single features. A rule is a tuple $< t, y >$ If a molecule contains substructure t, it is classified as y .

$$h_{<t,y>}(\mathbf{x}) = \begin{cases} y & \text{if } t \subseteq \mathbf{x}, \\ -y & otherwise. \end{cases}$$

  - Gain

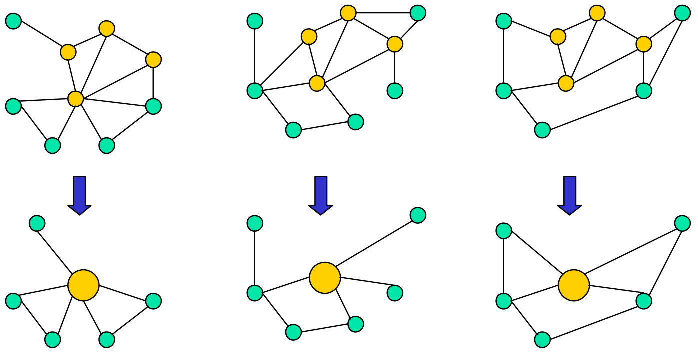
$$gain(< t, y >) = \sum_{i=1}^{n} y_i h_{<t,y>}(\mathbf{x}_i)$$

  - Applying boosting

$$gain(< t, y >) = \sum_{i=1}^{n} y_i \boxed{d_i} h_{<t,y>}(\mathbf{x}_i)$$

Graph Compression

- Extract common subgraphs and simplify graphs by condensing these subgraphs into nodes

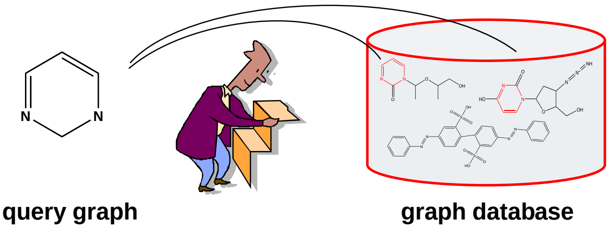


Outline

- Scalable pattern mining in graph data sets
  - Frequent subgraph pattern mining
  - Constraint-based graph pattern mining
  - Graph clustering, classification, and compression
- Searching graph databases
  - Graph indexing methods
  - Similarity search in graph databases
- Application and exploration with graph mining
  - Biological and social network analysis
  - Mining computer systems: bug isolation & performance tuning
- Conclusions and future work

Graph Search

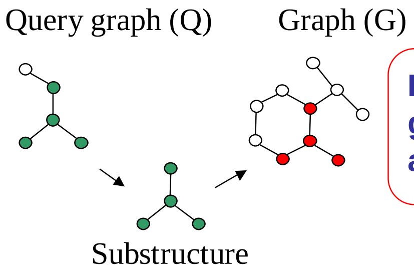
- Querying graph databases:
  - Given a graph database and a query graph, find all graphs containing this query graph



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

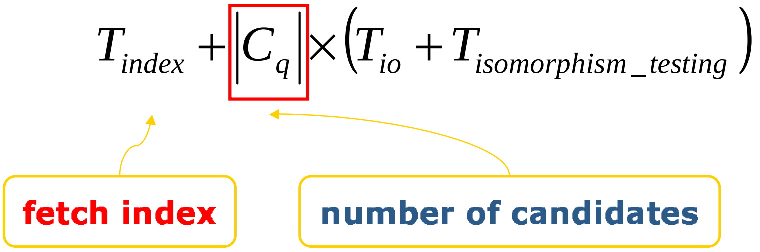
Step 2. Query Processing

- 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

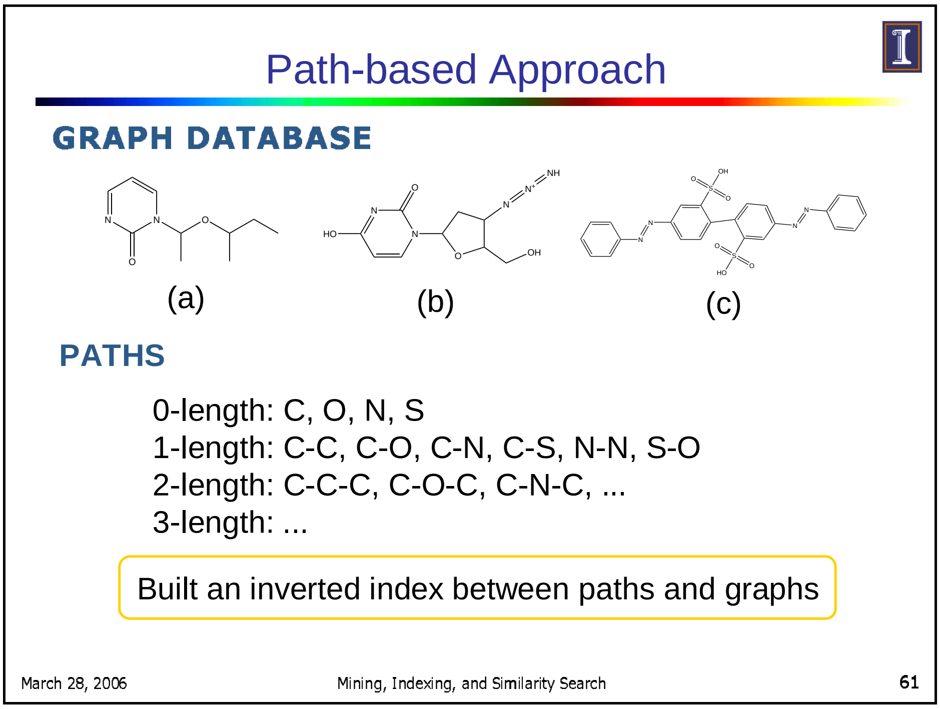


$$T_{index} + \boxed{\left|C_q\right|} \times \left(T_{io} + T_{isomorphism\_testing}\right)$$

fetch index

number of candidates

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



Path-based Approach

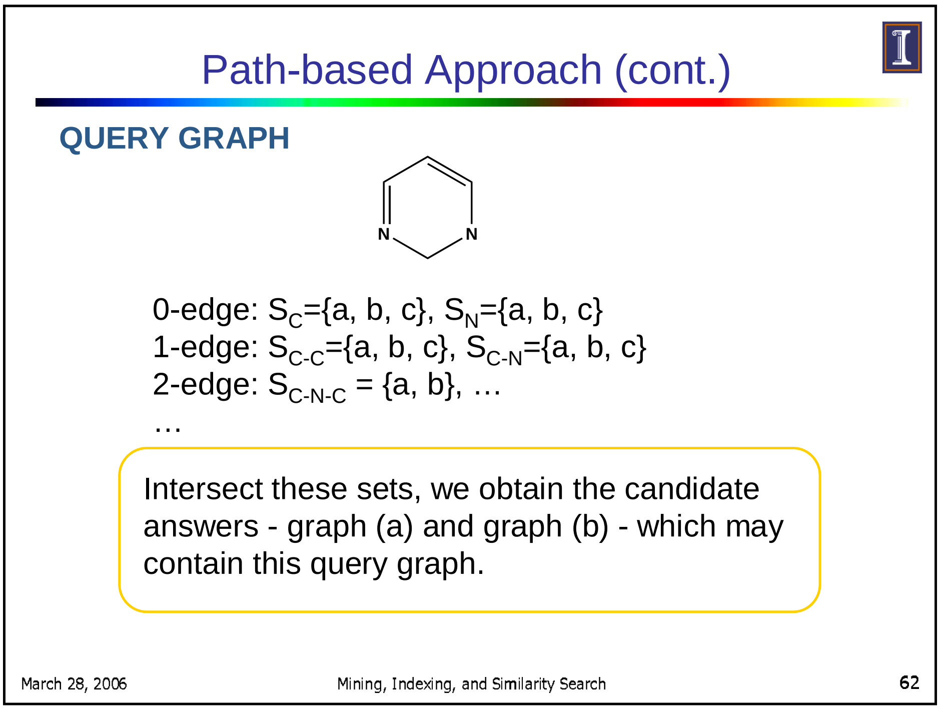
GRAPH DATABASE

(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

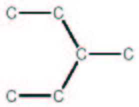
0-edge: S_C={a, b, c}, S_N={a, b, c}
1-edge: $S_{C\text{-}C}$={a, b, c}, $S_{C\text{-}N}$={a, b, c}
2-edge: $S_{C\text{-}N\text{-}C}$ = {a, b}, …
…

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

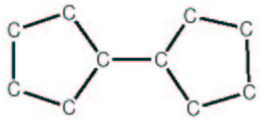
Problems: Path-based Approach

GRAPH DATABASE

(a)



(b)



(c)

QUERY GRAPH

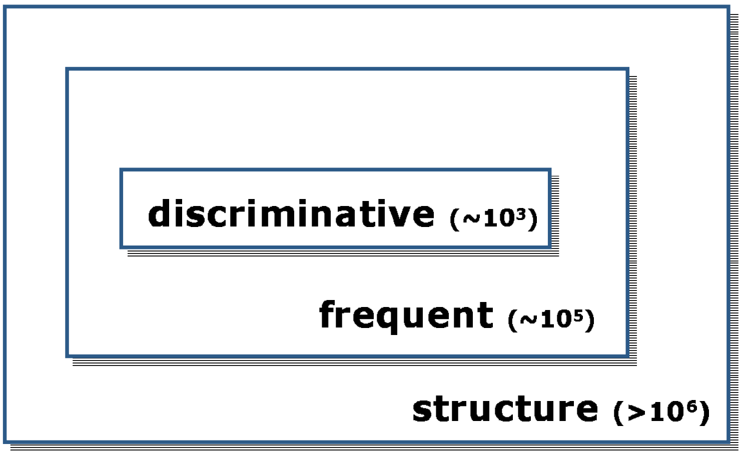
Only graph (c) contains this query graph. However, if we only index paths: C, C-C, C-C-C, C-C-C-C, we cannot prune graph (a) and (b).

---

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



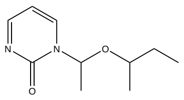
discriminative (~10^3)

frequent (~10^5)

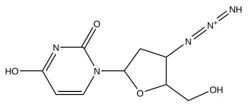
structure (>10^6)

Why Discriminative Subgraphs?

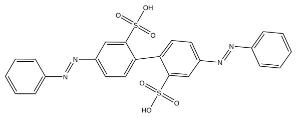
Sample database



(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 sturctures $f_1, f_2, \ldots f_n$ and a new structure x, we measure the extra indexing power provided by x,

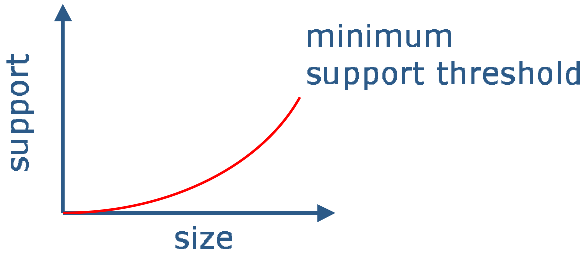
$$P(x | f_1, f_2, \ldots f_n),\ f_i \subset x.$$

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?

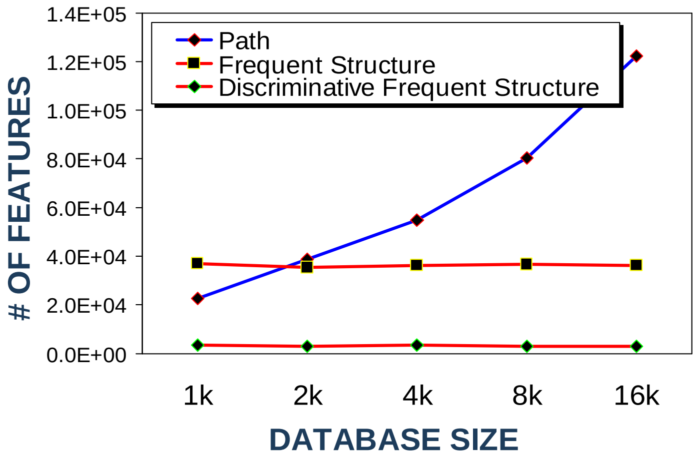
- We cannot index (or even search) all of substructures
- Large structures will likely be indexed well by their substructures
- Size-increasing support threshold

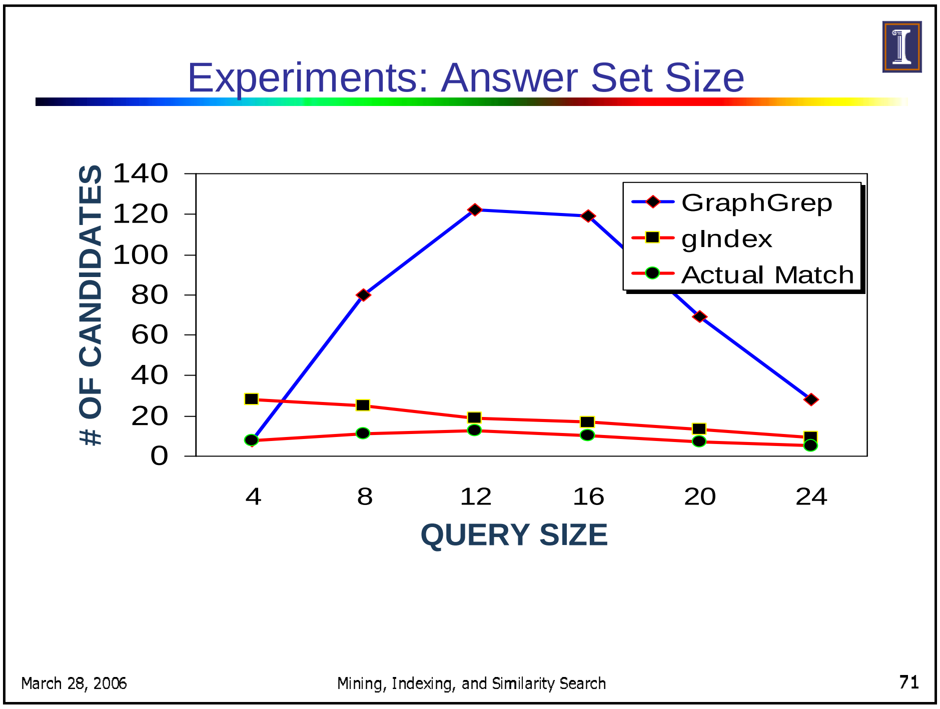


Experimental Setting

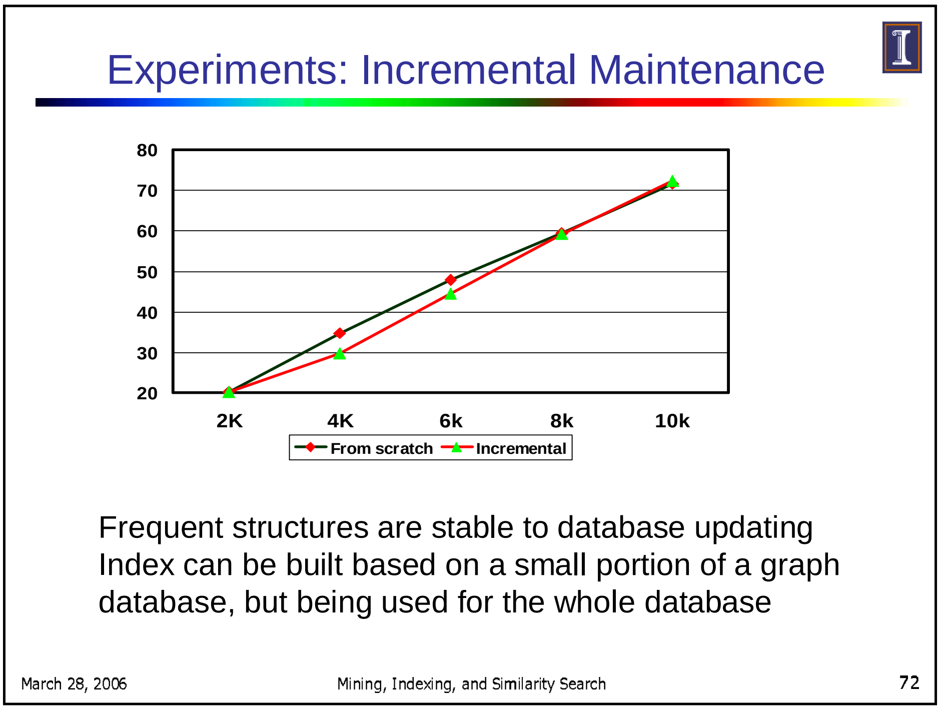
- The AIDS antiviral screen compound dataset from NCI/NIH, containing 43,905 chemical compounds
- Query graphs are randomly extracted from the dataset.
- GraphGrep: maximum length (edges) of paths is set at 10
- gIndex: maximum size (edges) of structures is set at 10

Experiments: Index Size





Experiments: Answer Set Size



Experiments: Incremental Maintenance

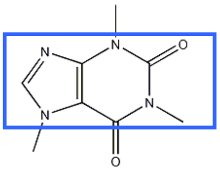
Frequent structures are stable to database updating
Index can be built based on a small portion of a graph database, but being used for the whole database

Outline

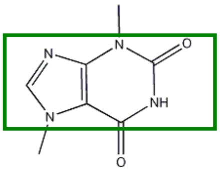
- Scalable pattern mining in graph data sets
  - Frequent subgraph pattern mining
  - Constraint-based graph pattern mining
  - Graph clustering, classification, and compression
- Searching graph databases
  - Graph indexing methods
  - Similarity search in graph databases
- Application and exploration with graph mining
  - Biological and social network analysis
  - Mining software systems: bug isolation & performance tuning
- Conclusions and future work

Structure Similarity Search

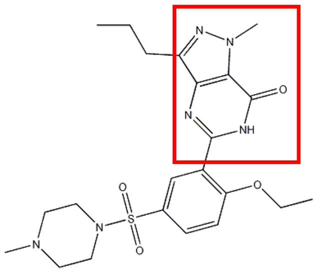
- **CHEMICAL COMPOUNDS**



(a) caffeine

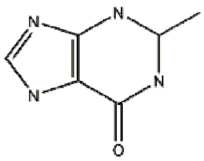


(b) diurobromine



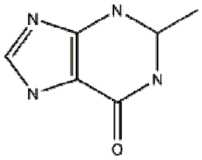
(c) viagra

- **QUERY GRAPH**



Some “Straightforward” Methods

- Method1: Directly compute the similarity between the graphs in the DB and the query graph
  - Sequential scan
  - Subgraph similarity computation



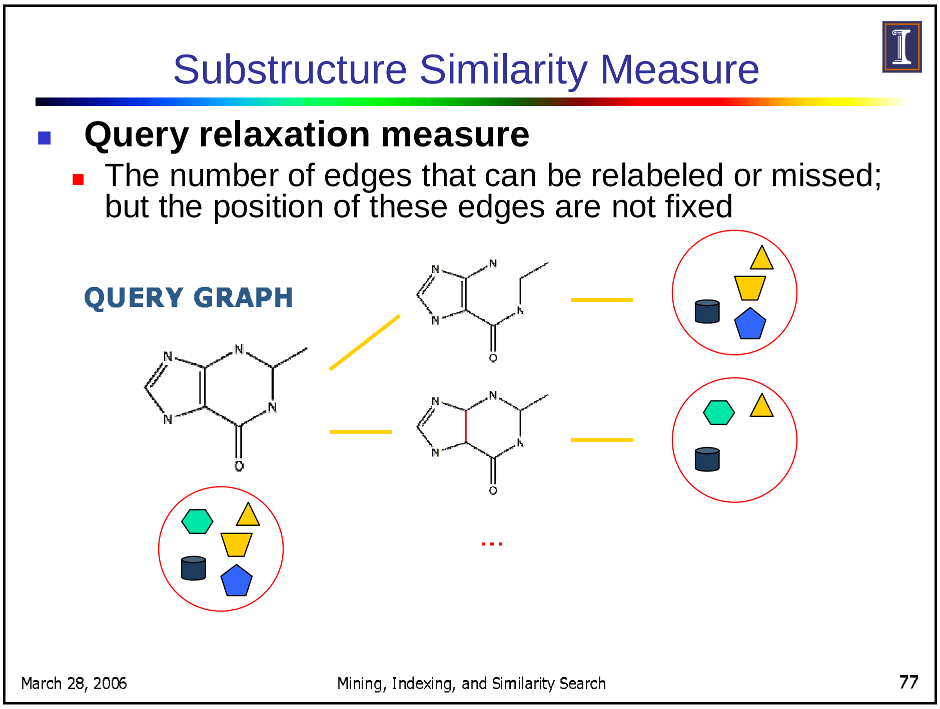
- Method 2: Form a set of subgraph queries from the original query graph and use the exact subgraph search
  - Costly: If we allow 3 edges to be missed in a 20-edge query graph, it may generate 1,140 subgraphs

---

Index: Precise vs. Approximate Search

- Precise Search
  - Use frequent patterns as indexing features
  - Select features in the **database space** based on their selectivity
  - Build the index
- Approximate Search
  - Hard to build indices covering similar subgraphs—explosive number of subgraphs in databases
  - Idea: (1) keep the index structure

    (2) select features in the **query space**



Substructure Similarity Measure

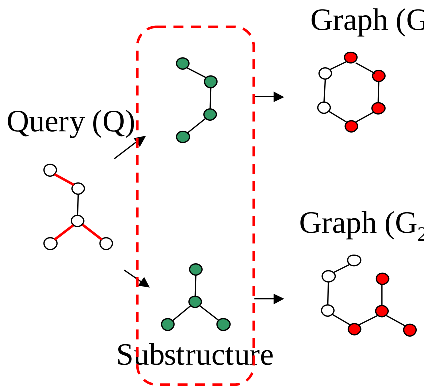
- **Query relaxation measure**
  - The number of edges that can be relabeled or missed; but the position of these edges are not fixed

QUERY GRAPH

Substructure Similarity Measure

- **Feature-based similarity measure**
  - Each graph is represented as a feature vector X = {x_1, x_2, …, x_n}
  - The similarity is defined by the distance of their corresponding vectors
  - Advantages
    - Easy to index
    - Fast
    - Rough measure

Intuition: Feature-Based Similarity Search



- If graph G contains the major part of a query graph Q, G should share a number of common features with Q

- Given a relaxation ratio, calculate the maximal number of features that can be missed !

At least one of them should be contained

Feature-Graph Matrix

graphs in database

features

	G_1	G_2	G_3	G_4	G_5
f_1	0	1	0	1	1
f_2	0	1	0	0	1
f_3	1	0	1	1	1
f_4	1	0	0	0	1
f_5	0	0	1	1	0
	✗	✗	✗		

Assume a query graph has 5 features and at most 2 features to miss due to the relaxation threshold

Edge Relaxation – Feature Misses

- If we allow k edges to be relaxed, J is the maximum number of features to be hit by k edges—it becomes the maximum coverage problem
- NP-complete
- A greedy algorithm exists

$$J_{\text{greedy}} \ge \left(1-\left(1-\frac{1}{k}\right)^{k}\right)\cdot J$$

  - We design a heuristic to refine the bound of feature misses

---

Query Processing Framework

- Three steps in processing approximate graph queries

Step 1. Index Construction

- Select small structures as features in a graph database, and build the feature-graph matrix between the features and the graphs in the database.

Framework (cont.)

Step 2. Feature Miss Estimation

- Determine the indexed features belonging to the query graph
- Calculate the upper bound of the number of features that can be missed for an approximate matching, denoted by J
    - On the query graph, not the graph database

Framework (cont.)

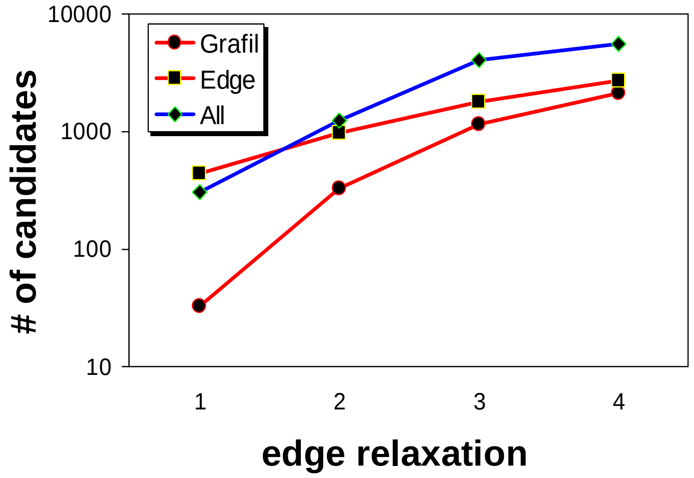
Step 3. Query Processing

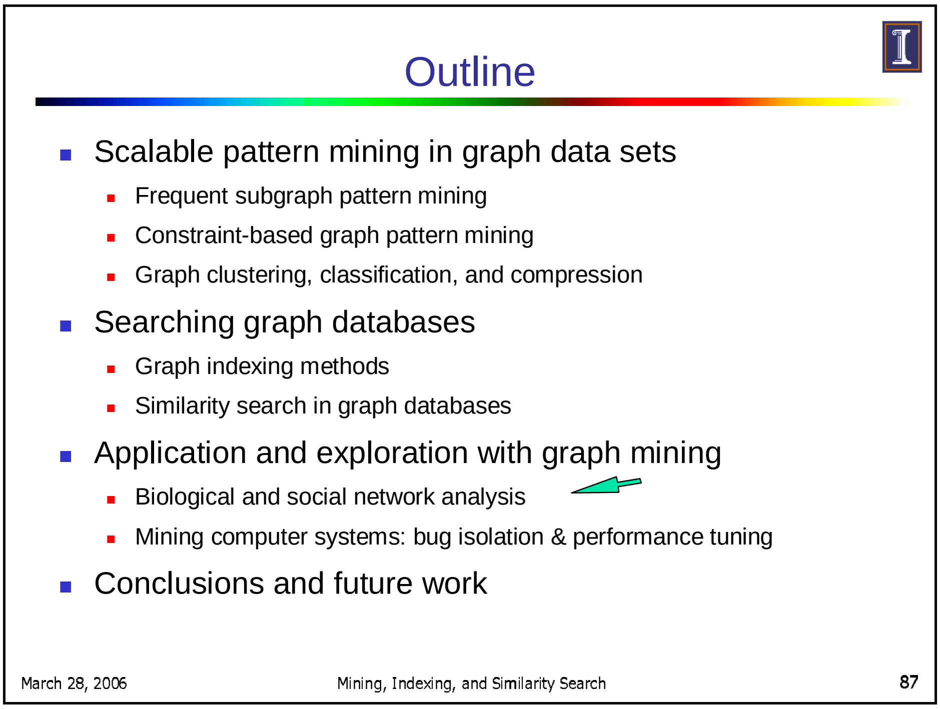
- Use the feature-graph matrix to calculate the difference in the number of features between graph G and query Q, $F_G – F_Q$
- If $F_G – F_Q > J$, discard G. The remaining graphs constitute a candidate answer set

Performance Study

- Database
  - Chemical compounds of Anti-Aids Drug from NCI/NIH, randomly select 10,000 compounds
- Query
  - Randomly select 30 graphs with 16 and 20 edges as query graphs
  - Competitive algorithms
    - Grafil: Graph Filter—our algorithm
    - Edge: use edges only
    - All: use all the features

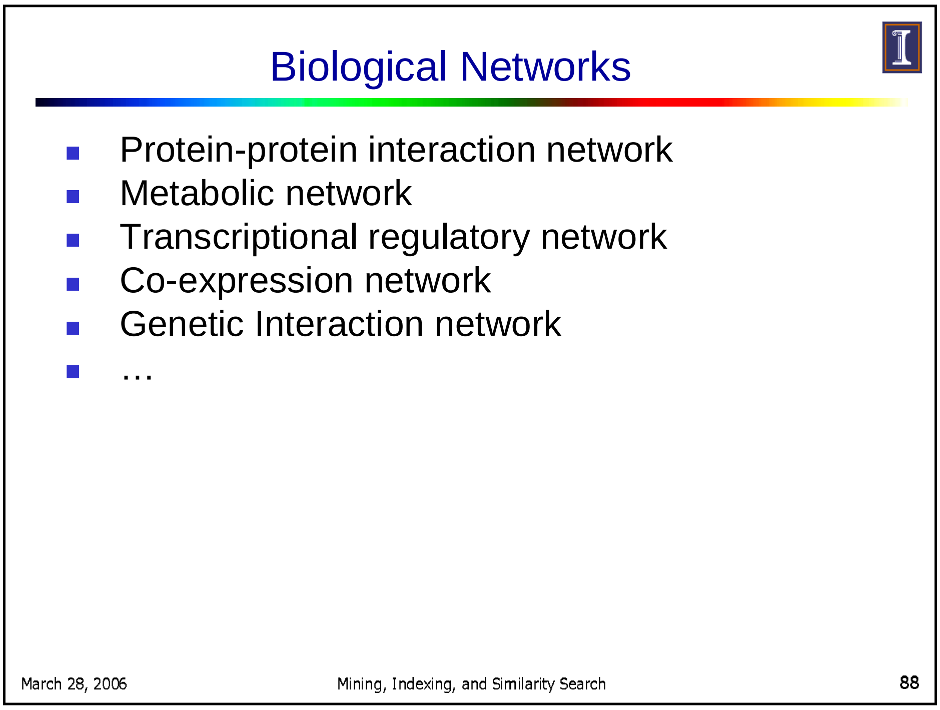
Comparison of the Three Algorithms





Outline

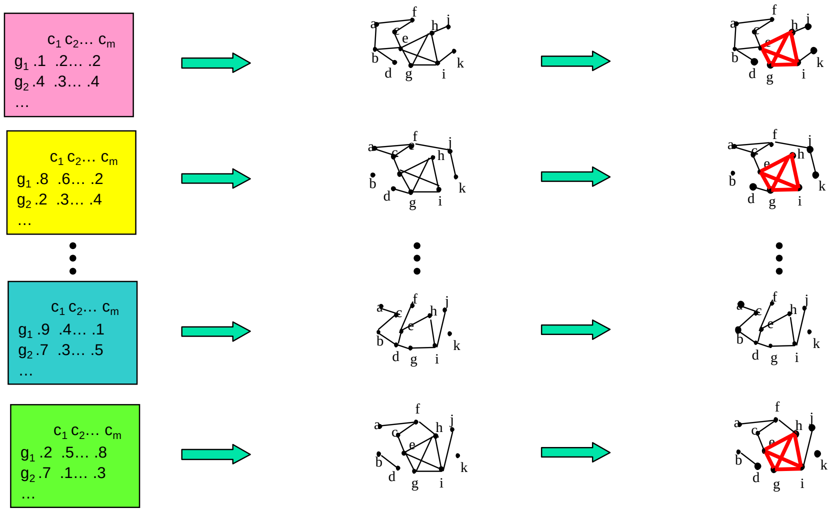
- Scalable pattern mining in graph data sets
  - Frequent subgraph pattern mining
  - Constraint-based graph pattern mining
  - Graph clustering, classification, and compression
- Searching graph databases
  - Graph indexing methods
  - Similarity search in graph databases
- Application and exploration with graph mining
  - Biological and social network analysis
  - Mining computer systems: bug isolation & performance tuning
- Conclusions and future work



Biological Networks

- Protein-protein interaction network
- Metabolic network
- Transcriptional regulatory network
- Co-expression network
- Genetic Interaction network
- …

Identify frequent co-expression clusters across multiple microarray data sets



c_1 c_2… c_m
g_1 .1 .2… .2
g_2 .4 .3… .4
…

c_1 c_2… c_m
g_1 .8 .6… .2
g_2 .2 .3… .4
…

c_1 c_2… c_m
g_1 .9 .4… .1
g_2 .7 .3… .5
…

c_1 c_2… c_m
g_1 .2 .5… .8
g_2 .7 .1… .3
…

Our Solution

We develop a novel algorithm, called *CODENSE*, to mine frequent ***coherent dense*** subgraphs.

The target subgraphs have three characteristics:

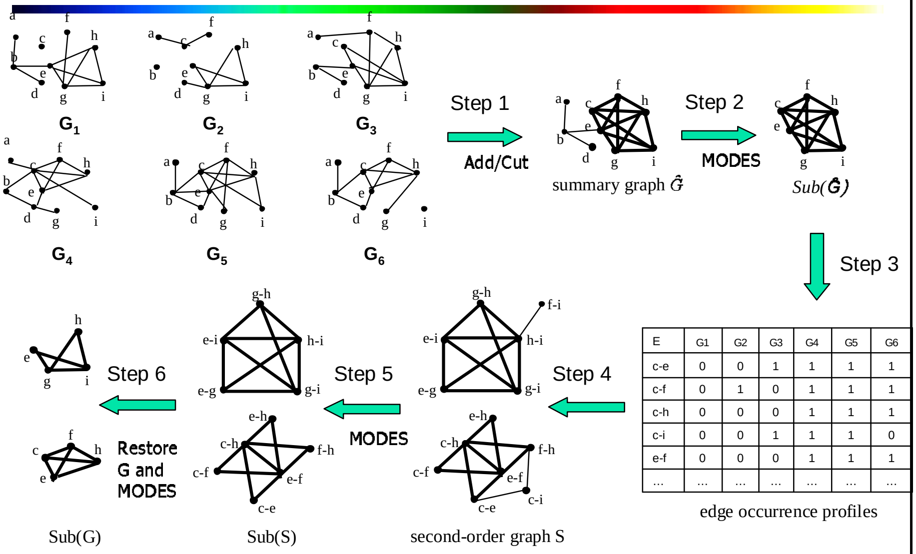
(1) All edges occur in >= k graphs (frequency)

(2) All edges should exhibit correlated occurrences in the given graph set (coherency)

(3) The subgraph is dense, where density *d* is higher than a threshold γ and $d=2m/(n(n-1))$ (density)

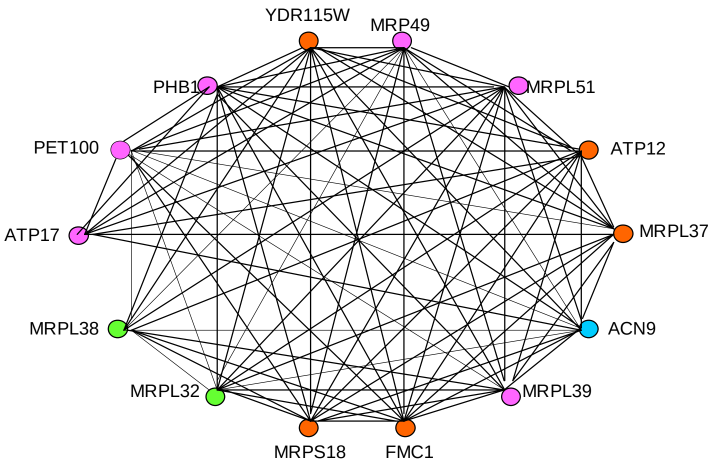
m: #edges, n: #nodes

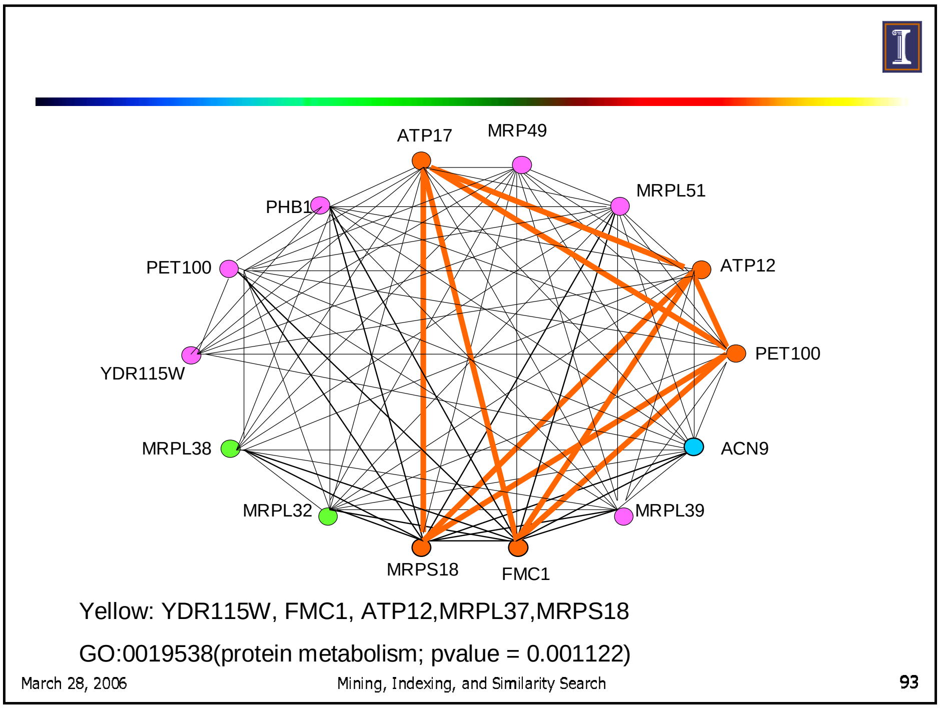
CODENSE: Mine coherent dense subgraphs



E	G1	G2	G3	G4	G5	G6
c-e	0	0	1	1	1	1
c-f	0	1	0	1	1	1
c-h	0	0	0	1	1	1
c-i	0	0	1	1	1	0
e-f	0	0	0	1	1	1
…	…	…	…	…	…	…

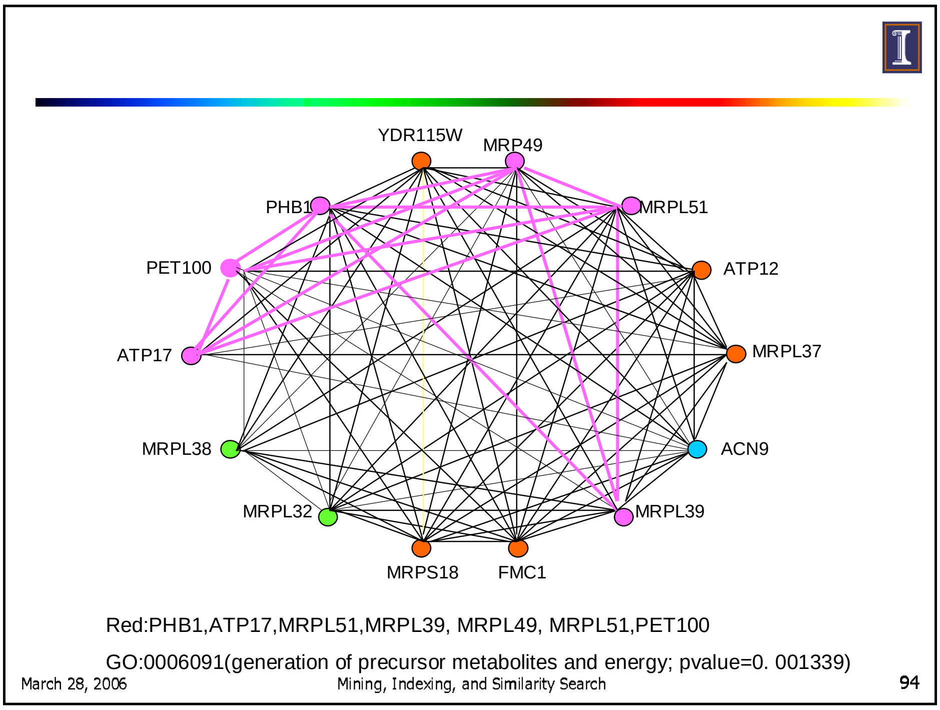
edge occurrence profiles





Yellow: YDR115W, FMC1, ATP12,MRPL37,MRPS18

GO:0019538(protein metabolism; pvalue = 0.001122)



Red:PHB1,ATP17,MRPL51,MRPL39, MRPL49, MRPL51,PET100

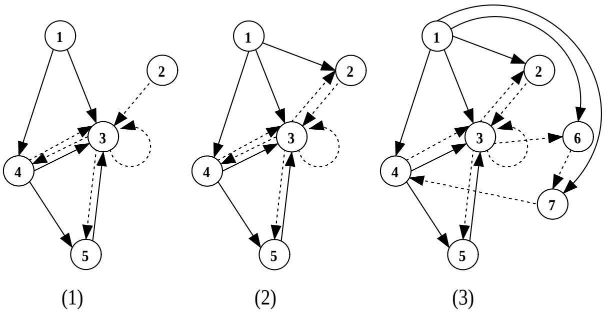
GO:0006091(generation of precursor metabolites and energy; pvalue=0. 001339)

Outline

- Scalable pattern mining in graph data sets
  - Frequent subgraph pattern mining
  - Constraint-based graph pattern mining
  - Graph clustering, classification, and compression
- Searching graph databases
  - Graph indexing methods
  - Similarity search in graph databases
- Application and exploration with graph mining
  - Biological and social network analysis
  - Mining computer systems: bug isolation & performance tuning
- Conclusions and future work

Bug Isolation by Program Flow Analysis

PROGRAM CALLER/CALLEE GRAPH



(1) (2) (3)

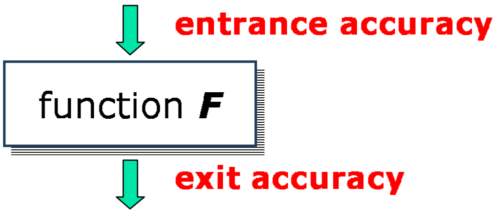
1: makepat
2: esc
3: addstr
4: getccl
5: dodash
6: in_set_2
7: stclose

Frequent Pattern-Based Classification

- Each program execution generates a (dynamic) caller/callee graph
- Extract frequent calling substructures from the correct and incorrect executions
- Use these substructures as features to classify

Watching the Boost of Classification Accuracy

- Bug detection based on the boost of classification accuracy
- Check the change of classification error at the entrance and at the exit of functions



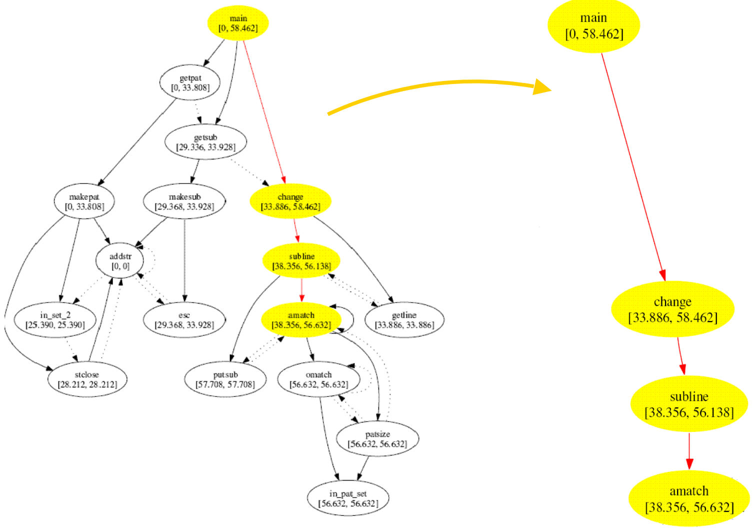
entrance accuracy

function ***F***

exit accuracy

- Compare their difference

Example: Bug Isolation by Data Mining



Outline

- Scalable pattern mining in graph data sets
  - Frequent subgraph pattern mining
  - Constraint-based graph pattern mining
  - Graph clustering, classification, and compression
- Searching graph databases
  - Graph indexing methods
  - Similarity search in graph databases
- Application and exploration with graph mining
  - Biological and social network analysis
  - Mining software systems: bug isolation & performance tuning
- Conclusions and future work

Conclusions

- Graph mining has wide applications
- Frequent and closed subgraph mining methods
  - gSpan and CloseGraph: pattern-growth depth-first search approach
- Graph indexing techniques
  - Frequent and discriminative subgraphs are high-quality indexing features
- Similarity search in graph databases
  - Indexing and feature-based matching
- Biological network analysis
  - Mining coherent, dense, multiple biological networks
- Program flow analysis

References (1)

- T. Asai, et al. “Efficient substructure discovery from large semi-structured data”, SDM'02
- C. Borgelt and M. R. Berthold, “Mining molecular fragments: Finding relevant substructures of molecules”, ICDM'02
- D. Cai, Z. Shao, X. He, X. Yan, and J. Han, “Community Mining from Multi-Relational Networks”, PKDD'05.
- M. Deshpande, M. Kuramochi, and G. Karypis, “Frequent Sub-structure Based Approaches for Classifying Chemical Compounds”, ICDM 2003
- M. Deshpande, M. Kuramochi, and G. Karypis. “Automated approaches for classifying structures”, BIOKDD'02
- L. Dehaspe, H. Toivonen, and R. King. “Finding frequent substructures in chemical compounds”, KDD'98
- C. Faloutsos, K. McCurley, and A. Tomkins, “Fast Discovery of 'Connection Subgraphs”, KDD'04
- H. Fröhlich, J. Wegner, F. Sieker, and A. Zell, “Optimal Assignment Kernels For Attributed Molecular Graphs”, ICML’05
- T. Gärtner, P. Flach, and S. Wrobel, “On Graph Kernels: Hardness Results and Efficient Alternatives”, COLT/Kernel’03
- L. Holder, D. Cook, and S. Djoko. “Substructure discovery in the subdue system”, KDD'94
- J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. “Mining spatial motifs from protein structure graphs”, RECOMB’04

References (2)

- J. Huan, W. Wang, and J. Prins. “Efficient mining of frequent subgraph in the presence of isomorphism”, ICDM'03
- H. Hu, X. Yan, Yu, J. Han and X. J. Zhou, ‶Mining Coherent Dense Subgraphs across Massive Biological Networks for Functional Discovery″, ISMB'05
- A. Inokuchi, T. Washio, and H. Motoda. “An apriori-based algorithm for mining frequent substructures from graph data”, PKDD'00
- C. James, D. Weininger, and J. Delany. “Daylight Theory Manual Daylight Version 4.82”. Daylight Chemical Information Systems, Inc., 2003.
- G. Jeh, and J. Widom, “Mining the Space of Graph Properties”, KDD'04
- H. Kashima, K. Tsuda, and A. Inokuchi, “Marginalized Kernels Between Labeled Graphs”, ICML’03
- M. Koyuturk, A. Grama, and W. Szpankowski. “An efficient algorithm for detecting frequent subgraphs in biological networks”, Bioinformatics, 20:I200--I207, 2004.
- T. Kudo, E. Maeda, and Y. Matsumoto, “An Application of Boosting to Graph Classification”, NIPS’04
- M. Kuramochi and G. Karypis. “Frequent subgraph discovery”, ICDM'01
- M. Kuramochi and G. Karypis, “GREW: A Scalable Frequent Subgraph Discovery Algorithm”, ICDM’04
- C. Liu, X. Yan, H. Yu, J. Han, and P. S. Yu, “Mining Behavior Graphs for ‘Backtrace" of Noncrashing Bugs"', SDM'05

References (3)

- P. Mahé, N. Ueda, T. Akutsu, J. Perret, and J. Vert, “Extensions of Marginalized Graph Kernels”, ICML’04
- B. McKay. Practical graph isomorphism. Congressus Numerantium, 30:45--87, 1981.
- S. Nijssen and J. Kok. A quickstart in frequent structure mining can make a difference. KDD'04
- J. Prins, J. Yang, J. Huan, and W. Wang. “Spin: Mining maximal frequent subgraphs from graph databases”. KDD'04
- D. Shasha, J. T.-L. Wang, and R. Giugno. “Algorithmics and applications of tree and graph searching”, PODS'02
- J. R. Ullmann. “An algorithm for subgraph isomorphism”, J. ACM, 23:31--42, 1976.
- N. Vanetik, E. Gudes, and S. E. Shimony. “Computing frequent graph patterns from semistructured data”, ICDM'02
- C. Wang, W. Wang, J. Pei, Y. Zhu, and B. Shi. “Scalable mining of large disk-base graph databases”, KDD'04
- T. Washio and H. Motoda, “State of the art of graph-based data mining”, SIGKDD Explorations, 5:59-68, 2003
- X. Yan and J. Han, ‶gSpan: Graph-Based Substructure Pattern Mining″, ICDM'02
- X. Yan and J. Han, ‶CloseGraph: Mining Closed Frequent Graph Patterns″, KDD'03

References (4)

- X. Yan, P. S. Yu, and J. Han, "Graph Indexing: A Frequent Structure-based Approach", SIGMOD'04
- X. Yan, X. J. Zhou, and J. Han, "Mining Closed Relational Graphs with Connectivity Constraints", KDD'05
- X. Yan, P. S. Yu, and J. Han, "Substructure Similarity Search in Graph Databases", SIGMOD'05
- X. Yan, F. Zhu, J. Han, and P. S. Yu, "Searching Substructures with Superimposed Distance", ICDE'06
- M. J. Zaki. "Efficiently mining frequent trees in a forest", KDD'02