Graph Mining Techniques and Their Applications

Sharma Chakravarthy
Information Technology Laboratory
Computer Science and Engineering Department
The University of Texas at Arlington, Arlington, TX 76009
Email: sharma@cse.uta.edu
URL: http://itlab.uta.edu/sharma

Tutorial Outline

- Data Mining Overview
- Need for Graph Mining
  - Applications
- Graph Mining Approaches
  - **Subdue**
  - **SQL-Based Subdue (HdbSubdue)**
  - **AGM**
  - ***FSG***
  - ***gSpan***
- Conclusions
- References

Motivation

Fraud division, some large telephone company:

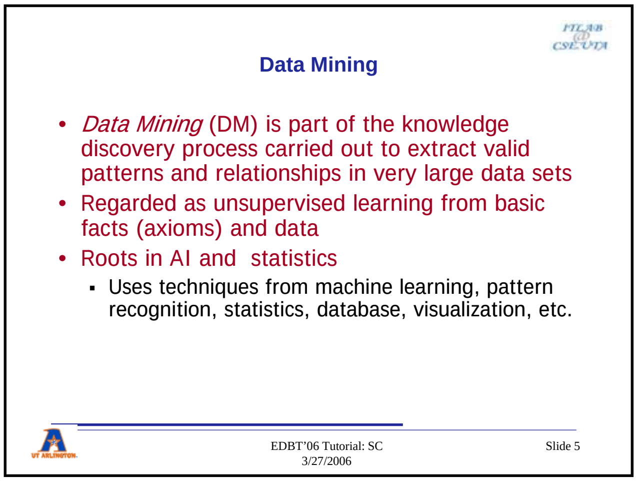
"How do we find these guys? There are 10 billion records on 10 million customers in the main database. With all this information we have about our customers and all the calls they make, can't you just ask the database to figure out which lines have been set-up *temporarily* and exhibited *similar* calling patterns in the *same time periods*? The information is in there, I just know it …"

Problem

- "Find-similar" problem just described is hard
  - e.g., "What products need to be improved?"
  - e.g., "Which books won't be checked out and can be taken off the shelves?"

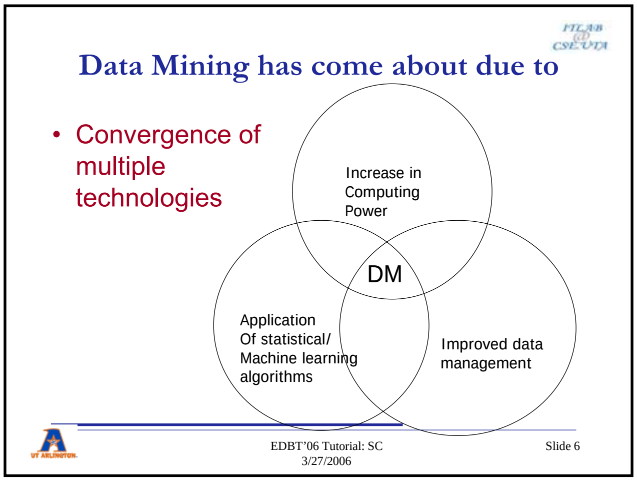
Why?

- Massive amounts of data
  - More and more online data stores (e.g., Web, click streams, corporate databases, etc.)
- No easy way to describe what to look for
- Traditional, interactive approaches fail
  - Size of data, different purposes



Data Mining

- *Data Mining* (DM) is part of the knowledge discovery process carried out to extract valid patterns and relationships in very large data sets
- Regarded as unsupervised learning from basic facts (axioms) and data
- Roots in AI and statistics
  - Uses techniques from machine learning, pattern recognition, statistics, database, visualization, etc.



Data Mining has come about due to

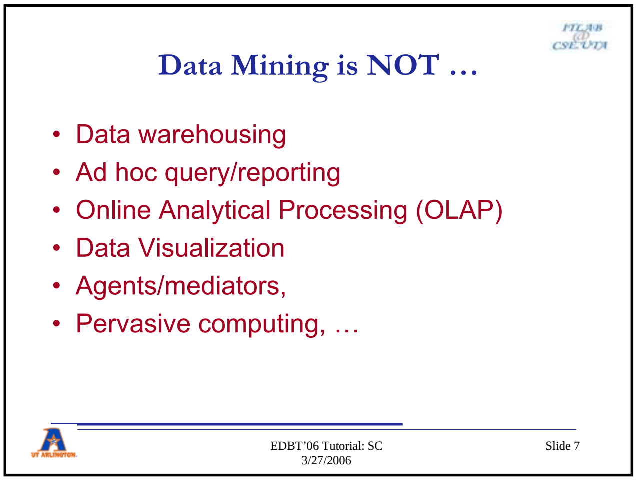
- Convergence of multiple technologies

Increase in Computing Power

DM

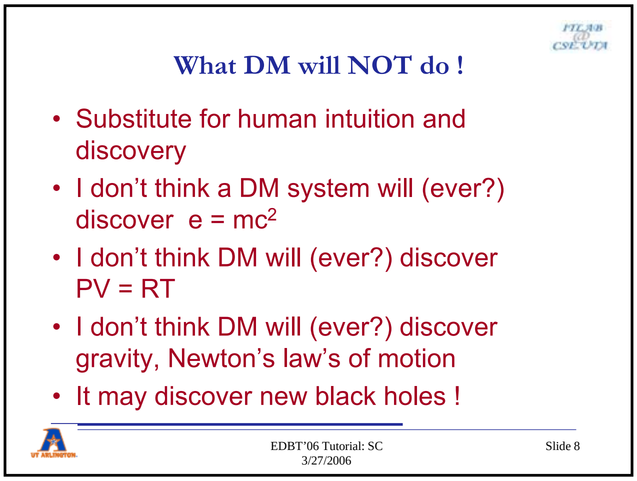
Application Of statistical/ Machine learning algorithms

Improved data management



Data Mining is NOT …

- Data warehousing
- Ad hoc query/reporting
- Online Analytical Processing (OLAP)
- Data Visualization
- Agents/mediators,
- Pervasive computing, …

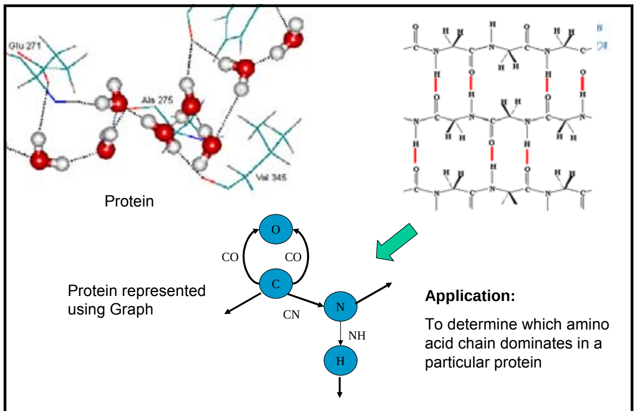


What DM will NOT do !

- Substitute for human intuition and discovery
- I don't think a DM system will (ever?) discover e = mc^2
- I don't think DM will (ever?) discover PV = RT
- I don't think DM will (ever?) discover gravity, Newton's law's of motion
- It may discover new black holes !

Need for Graph Mining

- Association Rule Mining, decision trees… mine transactional data.
- Graph based mining techniques are used for mining data that are structural in nature (chemical compounds, complex proteins, VLSI circuits, social networks, …) as mapping them to other representations is not possible or will lead to the loss of structural information
- Significant work in the area includes the Subdue substructure discovery algorithm (Cook & Holder), HdbSubdue (chakrvarthy, Beera, Padmanabhan), the apriori graph mining (AGM) (Inokuchi, Washio, and Motoda), the frequent subgraph (FSG) technique (Karypis & Kuramochi), and the gSpan approach (J. Han) (also SPIN (Huan, Wang, Prins, and Yang))



Protein

Protein represented using Graph

Application:

To determine which amino acid chain dominates in a particular protein

Application Domains

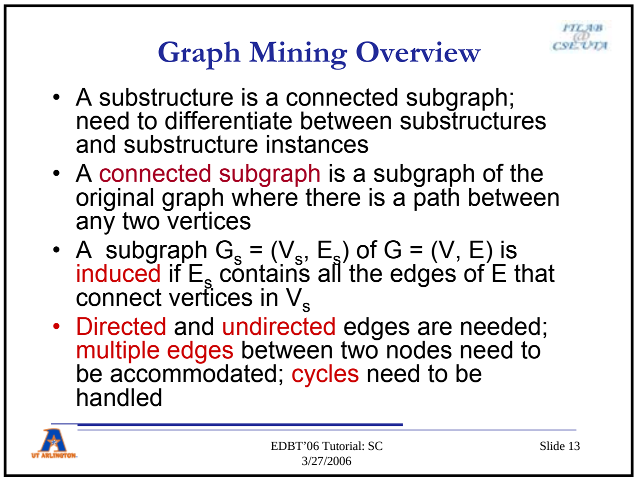
- Chemical Reaction chains
- CAD Circuit Analysis
- Social Networks
- Credit Domains
- Web analysis
- Games (Chess, Tic Tac toe)
- Program Source Code analysis
- Chinese Character data bases
- Geology
- Aviation Data Bases

Graph Based Data Mining

- A Graph representation of the database is intuitive and an obvious choice.
- Graphs can be used to accurately model and represent scientific data sets. Graphs are suitable for capturing arbitrary relations between the various objects.

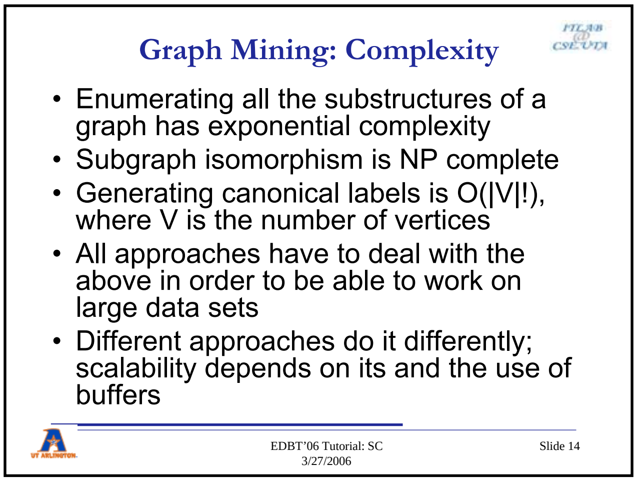
Data Instance	Graph Instance
Object	Vertex
Object's Attributes	Vertex Label
Relation Between Two Objects	Edge
Type Of Relation	Edge Label

- Graph based data mining aims at discovering interesting and repetitive patterns within these structural representations of data.



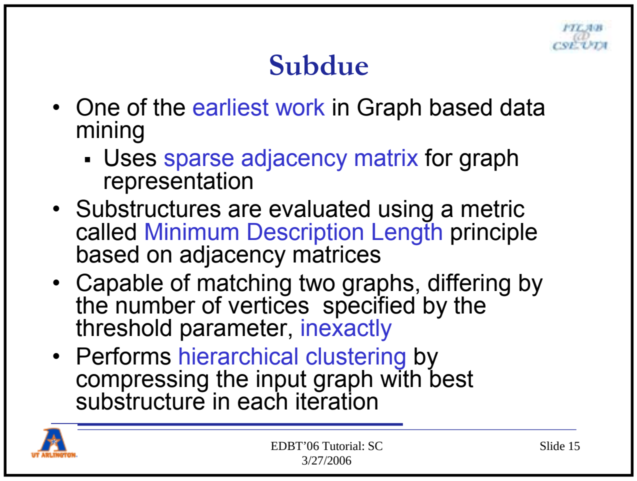
Graph Mining Overview

- A substructure is a connected subgraph; need to differentiate between substructures and substructure instances
- A connected subgraph is a subgraph of the original graph where there is a path between any two vertices
- A subgraph $G_s = (V_s, E_s)$ of $G = (V, E)$ is induced if E_s contains all the edges of E that connect vertices in V_s
- Directed and undirected edges are needed; multiple edges between two nodes need to be accommodated; cycles need to be handled



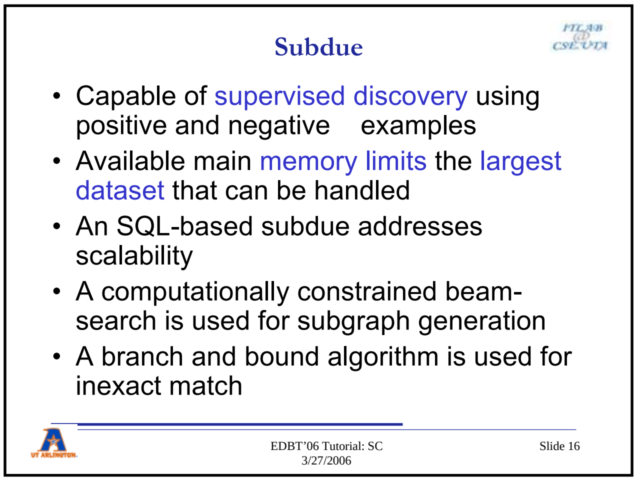
Graph Mining: Complexity

- Enumerating all the substructures of a graph has exponential complexity
- Subgraph isomorphism is NP complete
- Generating canonical labels is O(|V|!), where V is the number of vertices
- All approaches have to deal with the above in order to be able to work on large data sets
- Different approaches do it differently; scalability depends on its and the use of buffers



Subdue

- One of the earliest work in Graph based data mining
  - Uses sparse adjacency matrix for graph representation
- Substructures are evaluated using a metric called Minimum Description Length principle based on adjacency matrices
- Capable of matching two graphs, differing by the number of vertices specified by the threshold parameter, inexactly
- Performs hierarchical clustering by compressing the input graph with best substructure in each iteration



Subdue

- Capable of supervised discovery using positive and negative examples
- Available main memory limits the largest dataset that can be handled
- An SQL-based subdue addresses scalability
- A computationally constrained beam-search is used for subgraph generation
- A branch and bound algorithm is used for inexact match

AGM

- First to propose apriori-type algorithm for graph mining
- Detects frequent induced subgraphs for a given support
- Follows apriori algorithm
- Not much optimization; hence performance is not that good and is not scalable!

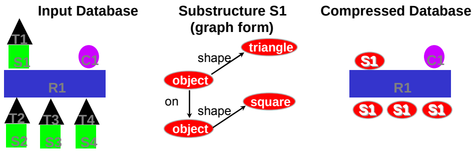
FSG

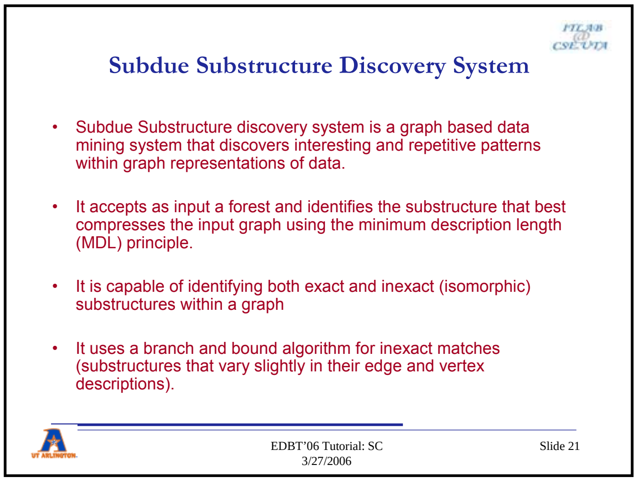
- FSG is used for frequent subgraph discovery
- Given a graph dataset *G* = *{G*1*,G*2*,G*3*, · · ·}*, it discovers all connected subgraphs that are found in at least the support threshold percent of the input graphs
- Uses a (sparse) adjacency matrix for graph representation
- A canonical label is generated by flattening the adjacency matrix of a graph (optimization)
- At each iteration FSG generates candidate subgraphs by adding one edge to the previous iteration's frequent subgraph (optimization)
- Graph isomorphism is checked by comparing canonical labels (optimization)

gSpan

- Avoids candidate generation
- Builds a new lexicographical ordering among graphs and maps each graph to a unique minimum DFS code as its canonical label
- Seems to outperform FSG
- Amenable to parallelization
- Does not handle cycles and multiple edges

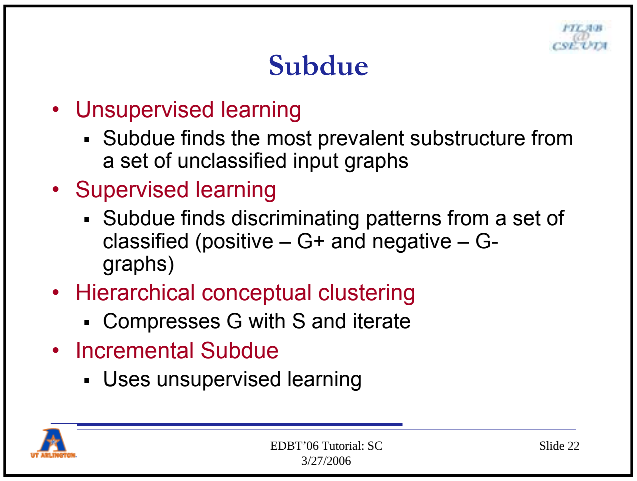
Subdue Example





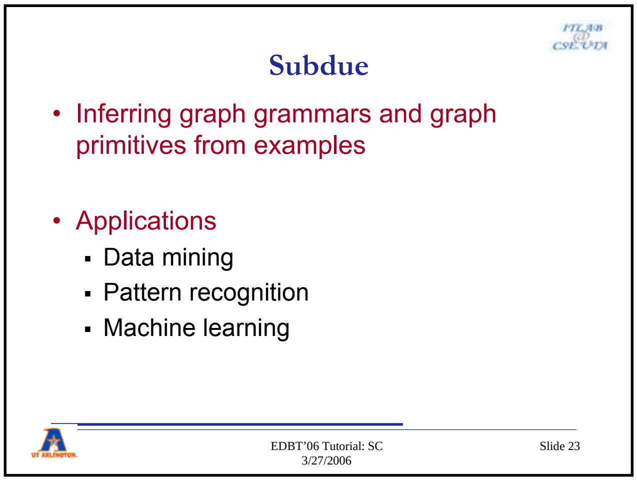
Subdue Substructure Discovery System

- Subdue Substructure discovery system is a graph based data mining system that discovers interesting and repetitive patterns within graph representations of data.
- It accepts as input a forest and identifies the substructure that best compresses the input graph using the minimum description length (MDL) principle.
- It is capable of identifying both exact and inexact (isomorphic) substructures within a graph
- It uses a branch and bound algorithm for inexact matches (substructures that vary slightly in their edge and vertex descriptions).



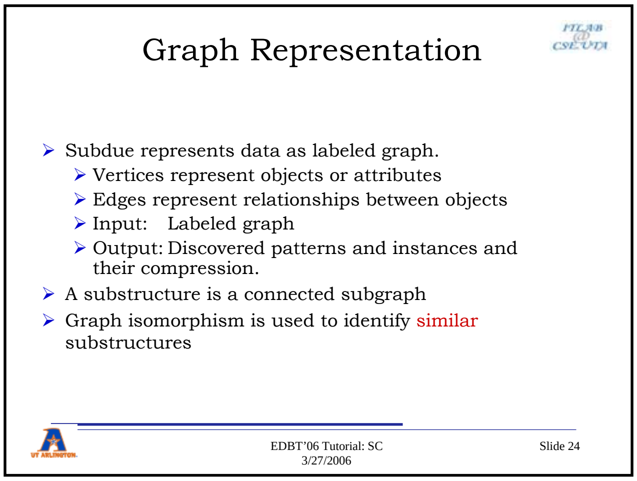
Subdue

- Unsupervised learning
  - Subdue finds the most prevalent substructure from a set of unclassified input graphs
- Supervised learning
  - Subdue finds discriminating patterns from a set of classified (positive – G+ and negative – G- graphs)
- Hierarchical conceptual clustering
  - Compresses G with S and iterate
- Incremental Subdue
  - Uses unsupervised learning



Subdue

- Inferring graph grammars and graph primitives from examples
- Applications
  - Data mining
  - Pattern recognition
  - Machine learning



Graph Representation

- Subdue represents data as labeled graph.
  - Vertices represent objects or attributes
  - Edges represent relationships between objects
  - Input: Labeled graph
  - Output: Discovered patterns and instances and their compression.
- A substructure is a connected subgraph
- Graph isomorphism is used to identify similar substructures

MDL Principle

- Theory to minimize description length (DL) of data; information theoretic approach
- Has been shown to be good across domains
- Evaluates substructures based on their ability to compress DL of graph
- Description length =DL(S) + DL(G/S)
  - Depends upon the representation
  - Substructure that best compresses the original is chosen

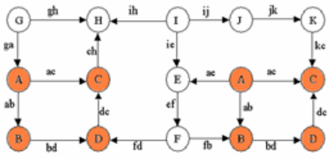
MDL Principle (cont.)

- Minimizes description length (DL) of data
- Substructures are evaluated based on their ability to compress the DL of the entire graph
- MDL = description length of the compressed graph / description length of the original graph

$$\text{MDL} = \frac{\text{DL(G)}}{\text{DL(S)} + \text{DL(G}\,|\,\text{S)}}$$

- DL(G) – Description length of the input graph
- DL(S) – Description length of sub graph
- DL(G|S) – Description length of the graph given the sub graph

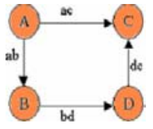
Example: Subdue



Input

- The input is a file, with all the vertex labels, vertex numbers, edges (using vertex numbers) and the edge directions

```
 v 1 A
v 2 B
v 3 C
v 4 D
d 1 2 ab
d 1 3 ac
d 2 4 bd
d 4 3 dc
```



- ‘d’ stands for a directed edge and ‘u’ stands for undirected. ‘e’ stands for directed unless specified as –undirected at the command prompt.

Subdue Approach

- Create a substructure for each unique vertex label
- Expand each substructure by adding an edge (and may be a vertex)
- Maintain **beam** number of substructures for expansion
- Halting conditions
  - Discovered substructures > **limit**
  - List maintaining the substructures to be expanded becomes empty
  - **Max size** of substructure to be discovered is reached

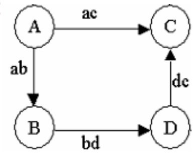
Output

- Output

  Substructure: MDL value = 1.21789, instances = 2

  Graph (4v,4e):

  v 1 A
  v 2 C
  v 3 B
  v 4 D
  d 1 2 ac
  d 1 3 ab
  d 4 2 dc
  d 3 4 bd



Subdue Parameters

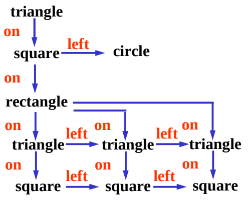
- *Threshold* determines the amount of variation permissible in the vertex and edge descriptions during inexact graph match.
- *Nsubs* determines the maximum number of substructures that are returned as the set of best substructures
- *Beam* determines the maximum number of substructures that are retained for expansion in the next iteration of the discovery algorithm
- Minsize constrains the size of substructures returned as best to be equal to or more than the specified parameter value

Algorithm

- SUBDUE(G, limit, beam)
  - S = vertices(G)
  - While (computation < limit) and (S <> {})
    - Order S from best to worst using MDL and background knowledge rules
    - S = first beam structures of S
    - b = first(S)
    - E = (b extended by one edge in all possible ways}
    - S = S U E
  - Return the substructure that produces the best compression ratio

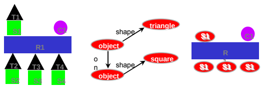
Algorithm (Contd.)

1. Create substructure for each unique vertex label



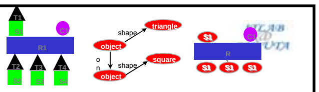
Substructures:

triangle (4), square (4), circle (1), rectangle (1)

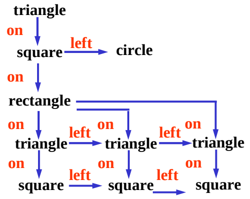


EDBT'06 Tutorial: SC
3/27/2006
Slide 33

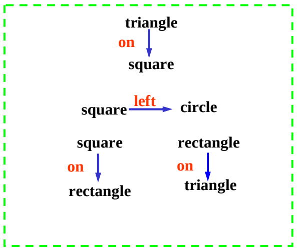
Algorithm (Contd.)



2. Expand best substructure by an edge or edge+neighboring vertex

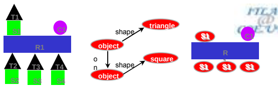


Substructures:



EDBT'06 Tutorial: SC
3/27/2006
Slide 34

Algorithm (cont.)



3. Keep only best substructures on queue (specified by beam width)
4. Terminate when queue is empty or #discovered substructures >= limit
5. Compress graph and repeat to generate hierarchical description

- Constrained to run in polynomial time

EDBT'06 Tutorial: SC
3/27/2006
Slide 35

Graph Match

- Exact Graph match
- Inexact Graph match

Exact graph match is likely to be restrictive for real life applications.

EDBT'06 Tutorial: SC
3/27/2006
Slide 36

Inexact Graph Match

- Some variations may occur between instances
- Want to abstract over minor differences
- Difference = cost of transforming one graph to make it isomorphic to another
- Match if cost/size < threshold

Inexact Graph Match

- Minimum graph edit distance

cumulative cost of graph changes required to transform the first graph into a graph isomorphic to the second graph.

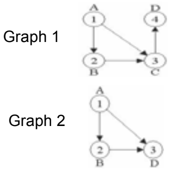
- Uniform Cost Search

- Exact graph match is NP complete
- Bunke and Allerman's approach

  Each distortion is assigned a cost.

  A distortion is a basic transformation such as deletion, insertion of vertices and edges (and their labels)

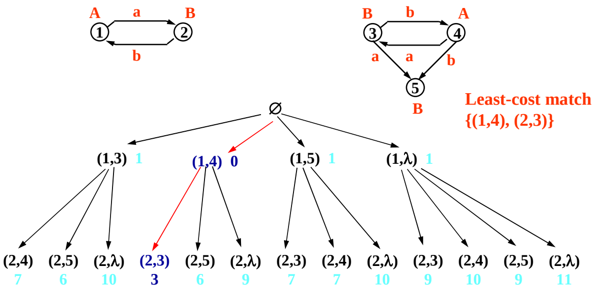
  Fuzzy graph match is a mapping f: $N_1 \rightarrow N_2 \cup \{\lambda\}$, N_1 and N_2 are sets of nodes of graph 1 and graph 2. A node v є N_1 that is mapped to λ is deleted
- If matchcost <= threshold then two graphs are said to be isomorphic
- Employing computational constraints such as bound on the number of substructures considered makes subdue run in polynomial time



Mapping : 1 → 1, 2 → 2, 3 → 3, 4 → null
1) Delete Edge 3 → 4 (cost = 2) edge + label
2) Delete vertex 4 (cost = 2) node + label
3) Substitute vertex label C by D for vertex 3 (cost = 1)

Total cost = 5 for this mapping

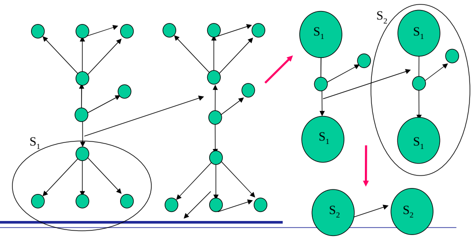
Inexact Graph Match



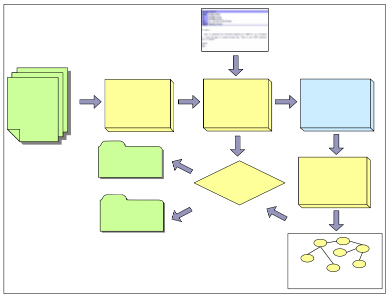
Least-cost match is {(1,4), (2,3)}

Hierarchical Reduction

- Input is a labeled graph
- A substructure is connected subgraph
- A substructure instance is a subgraph isomorphic to substructure definition
- Multiple iterations can create hierarchy



Document Classification Example



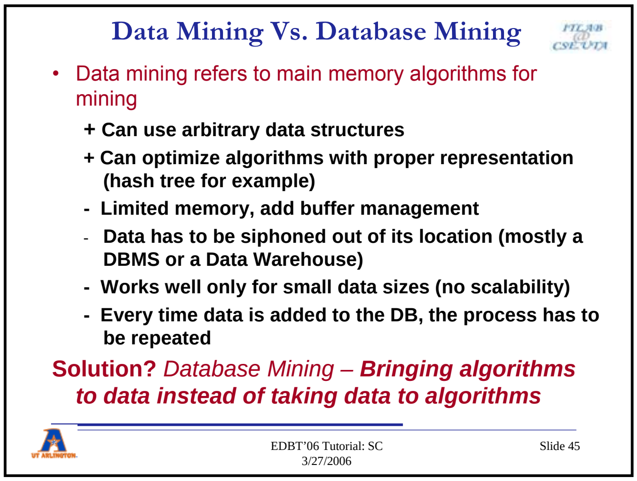
Control flow in the InfoSift classification system

Variants of Subdue

- Concept learner using positive and negative examples
- Hierarchical reduction
- Similarity detection in social networks
- Database approach to some of the above

Why Database Mining?

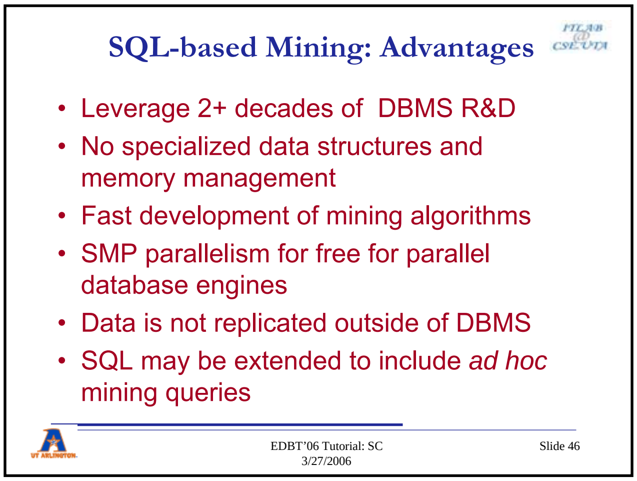
- Proliferation of relational DW and the need to mine them
- Data mining must ‵‵co-exist'' with OLAP and other decision-support applications
- DM will be a sub-process in next generation Business Intelligence (BI) Systems
- Leverage the RDBMS technology for mining
- Provide an integrated decision-support environment for analysts



Data Mining Vs. Database Mining

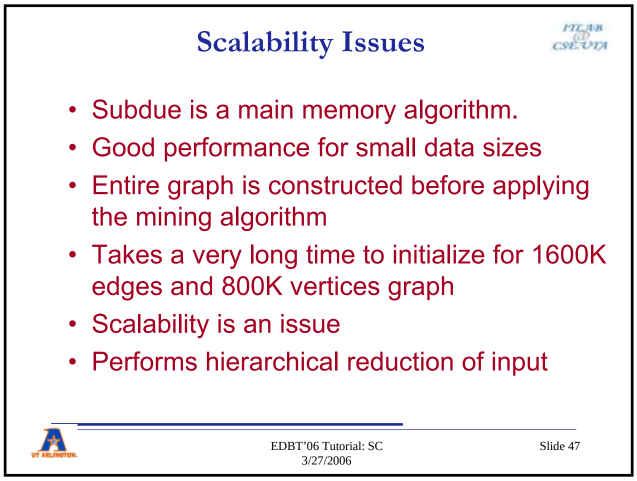
- Data mining refers to main memory algorithms for mining
  - **+ Can use arbitrary data structures**
  - **+ Can optimize algorithms with proper representation (hash tree for example)**
  - **- Limited memory, add buffer management**
  - **- Data has to be siphoned out of its location (mostly a DBMS or a Data Warehouse)**
  - **- Works well only for small data sizes (no scalability)**
  - **- Every time data is added to the DB, the process has to be repeated**

Solution? *Database Mining* – ***Bringing algorithms to data instead of taking data to algorithms***



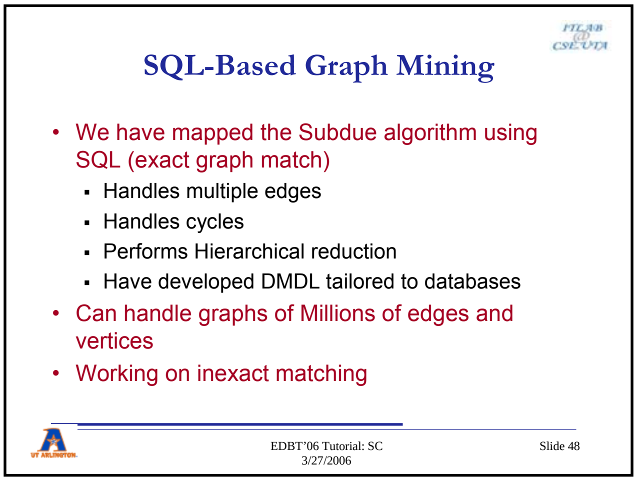
SQL-based Mining: Advantages

- Leverage 2+ decades of DBMS R&D
- No specialized data structures and memory management
- Fast development of mining algorithms
- SMP parallelism for free for parallel database engines
- Data is not replicated outside of DBMS
- SQL may be extended to include *ad hoc* mining queries



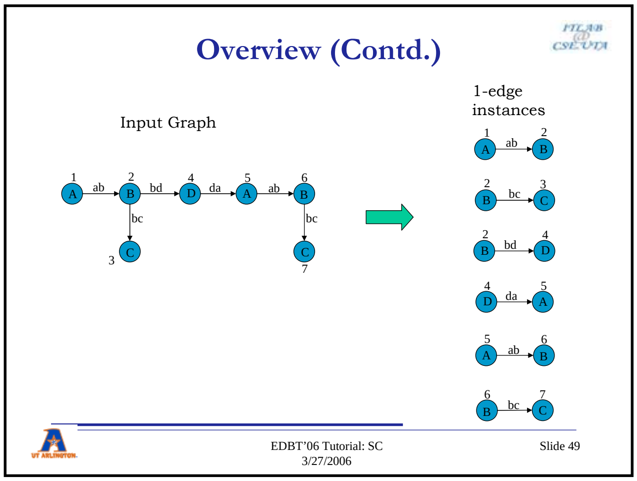
Scalability Issues

- Subdue is a main memory algorithm.
- Good performance for small data sizes
- Entire graph is constructed before applying the mining algorithm
- Takes a very long time to initialize for 1600K edges and 800K vertices graph
- Scalability is an issue
- Performs hierarchical reduction of input



SQL-Based Graph Mining

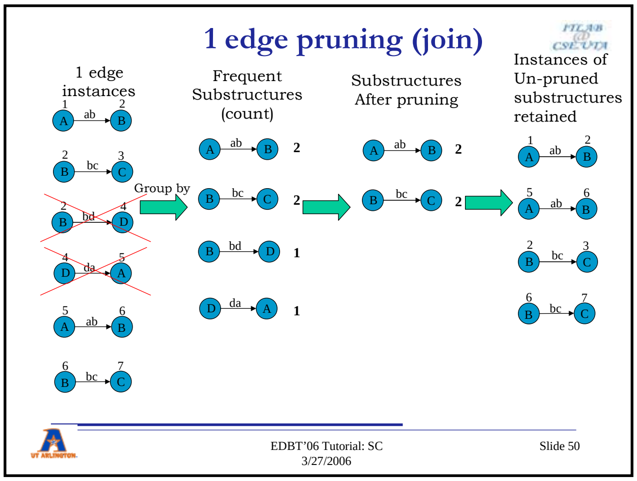
- We have mapped the Subdue algorithm using SQL (exact graph match)
  - Handles multiple edges
  - Handles cycles
  - Performs Hierarchical reduction
  - Have developed DMDL tailored to databases
- Can handle graphs of Millions of edges and vertices
- Working on inexact matching



Overview (Contd.)

Input Graph

1-edge instances



1 edge pruning (join)

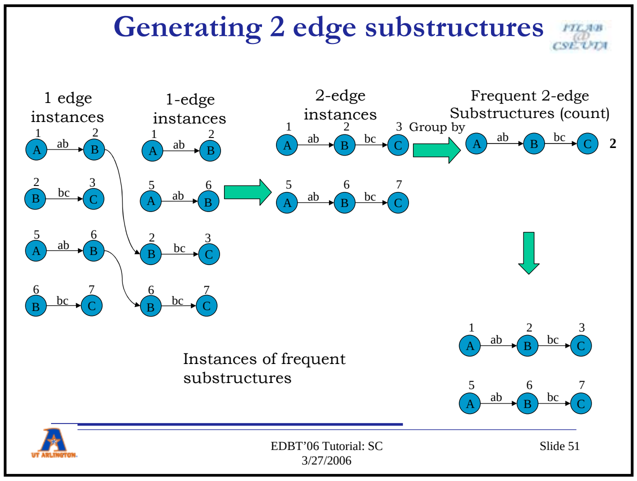
1 edge instances

Frequent Substructures (count)

Substructures After pruning

Instances of Un-pruned substructures retained

Group by



Generating 2 edge substructures

1 edge instances

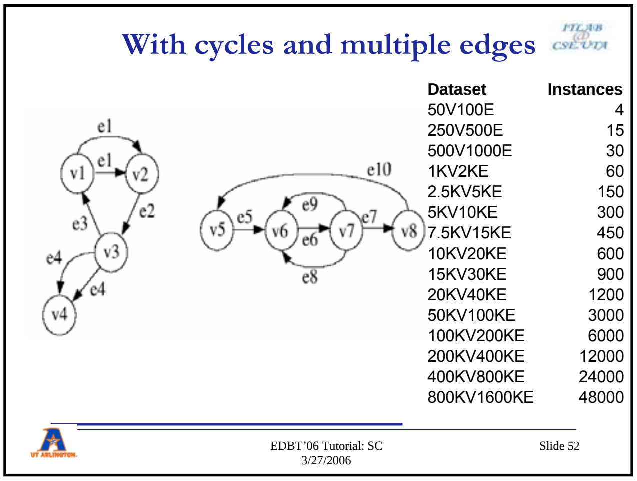
1-edge instances

2-edge instances

Frequent 2-edge Substructures (count)

Group by

Instances of frequent substructures

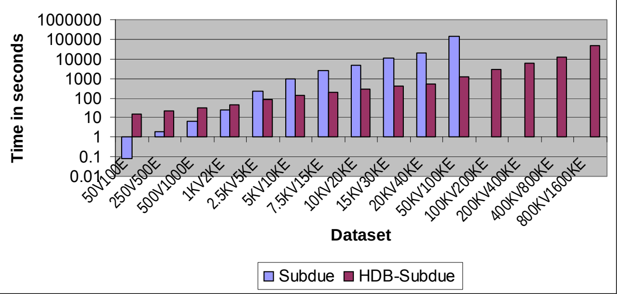


With cycles and multiple edges

Dataset	Instances
50V100E	4
250V500E	15
500V1000E	30
1KV2KE	60
2.5KV5KE	150
5KV10KE	300
7.5KV15KE	450
10KV20KE	600
15KV30KE	900
20KV40KE	1200
50KV100KE	3000
100KV200KE	6000
200KV400KE	12000
400KV800KE	24000
800KV1600KE	48000

With cycles and multiple edges

Beam 4, MaxSize 5, Iterations 1

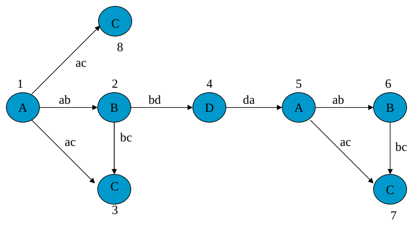


- Subdue crossover – 2.5KV5KE

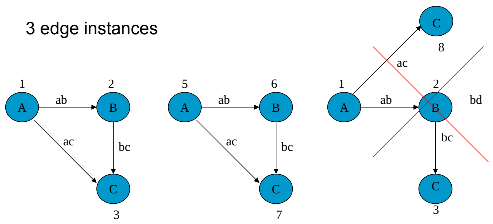
HdbSubdue

- Graph representation using relations
- Joins used for iterative generation of larger substructures
- Pseudo duplicate elimination involves a number of joins
- DMDL is used to identify the best substructure (count/frequency can be used as well)

Representation

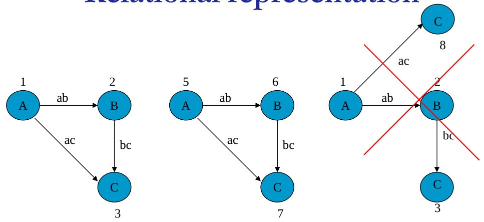


3 edge instances



Isomorphic instance Count 2

Relational representation



HDB instance_3 (instances of size 3)

V1	V2	V3	V4	V1L	V2L	V3L	V4L	E1	E2	E3	F1	T1	F2	T2	F3	T3
1	2	3	0	A	B	C	-	ab	bc	ac	1	2	2	3	1	3
5	6	7	0	A	B	C	-	ab	bc	ac	1	2	2	3	1	3
1	2	3	8	A	B	C	C	ab	bc	ac	1	2	2	3	1	8

Count 3

AGM

- Apriori-based Graph Mining
  - Combines adjacency matrix with the efficient level-wise search of the frequent canonical matrix code
  - Adjacency matrix elements are numbers (e.g., edge numbers) instead of being binary
  - Support and confidence are redefined for the graph domain

AGM (Contd.)

- The subset property is preserved by using normal forms of adjacency matrix
- If the adjacency matrix generated is not in the normal form, it has to be transformed to the normal form.
- Support counting is done on the database as in an apriori algorithm.
- An efficient indexing is used for this purpose

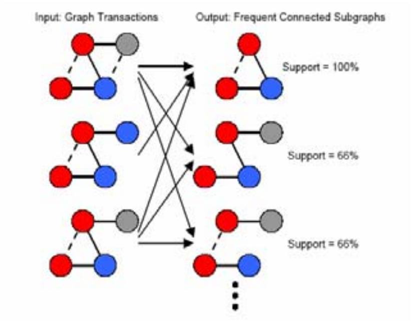
FSG

- Aims at discovering interesting sub-graph(s) that appear frequently over the entire set of graphs in contrast to discovering a interesting sub-graph(s) that appear within a single graph (or a forest) as in Subdue/HDB-Subdue
- It is designed along the lines of Apriori algorithm.

Problem Definition

- discovering all connected subgraphs that occur frequently over the entire set of graphs.
  - Subdue: best n are output (n is user defined)
- vertex : correspond to an entity
- edge : correspond to a relation between two entities

Example of Frequent sub-graph discovery

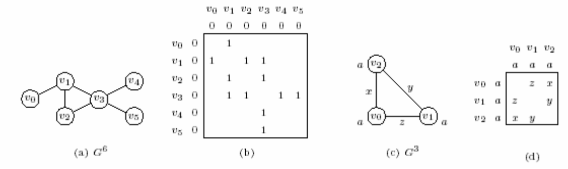

Input: Graph Transactions
Output: Frequent Connected Subgraphs
Support = 100%
Support = 66%
Support = 66%

Key Features Of FSG

Uses sparse graph representation that minimizes storage and computation
- (Subdue does the same)

- Increases the size of frequent subgraphs by adding one edge at a time (apriori)
  - (Subdue does the same)
- Uses canonical labeling to uniquely identify subgraphs
  - (Subdue uses bounded subgraph-isomorphism)
- ONLY undirected edges; I believe it cannot handle multiple edges and cycles
  - Unlike subdue

Canonical Labeling



(a) G^6

$$\begin{array}{cc|cccccc} & & v_0 & v_1 & v_2 & v_3 & v_4 & v_5 \\ & & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline v_0 & 0 & & 1 & & & & \\ v_1 & 0 & 1 & & 1 & 1 & & \\ v_2 & 0 & & 1 & & 1 & & \\ v_3 & 0 & & 1 & 1 & & 1 & 1 \\ v_4 & 0 & & & & 1 & & \\ v_5 & 0 & & & & 1 & & \end{array}$$

(b)

(c) G^3

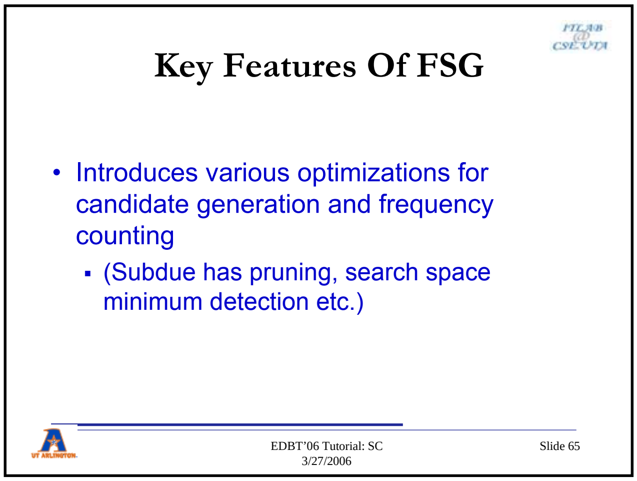
$$\begin{array}{cc|ccc} & & v_0 & v_1 & v_2 \\ & & a & a & a \\ \hline v_0 & a & & z & x \\ v_1 & a & z & & y \\ v_2 & a & x & y & \end{array}$$

(d)

"000000 1 01 011 0001 00010"

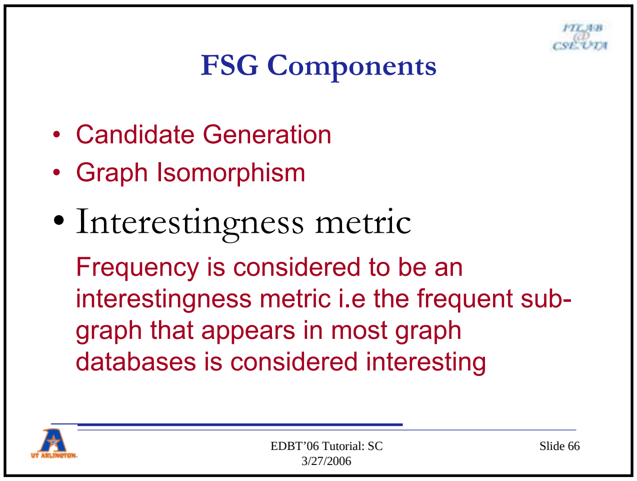
"*aaa z xy*"

- Different orderings of the vertices will give rise to different codes
- Try every possible permutation of the vertices and choose the ordering which gives lexicographically the largest, or the smallest code.
- O(|V|!)



Key Features Of FSG

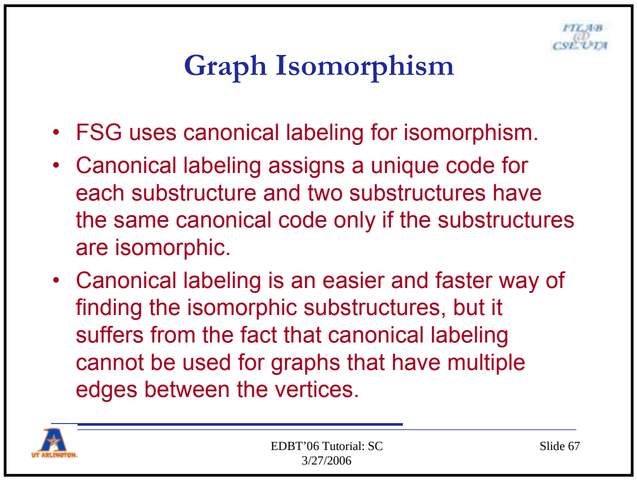
- Introduces various optimizations for candidate generation and frequency counting
  - (Subdue has pruning, search space minimum detection etc.)



FSG Components

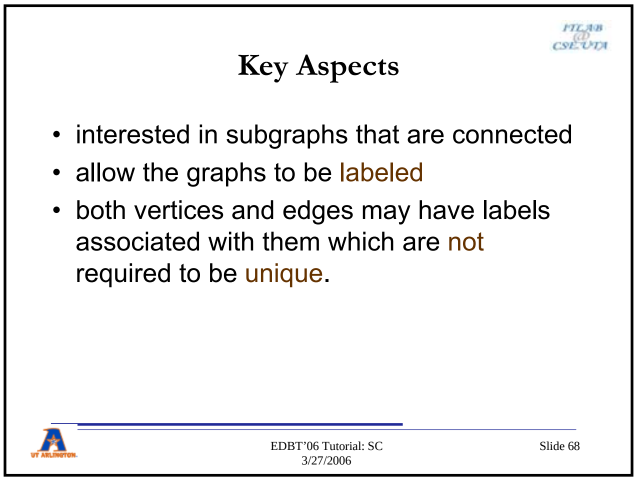
- Candidate Generation
- Graph Isomorphism
- Interestingness metric

  Frequency is considered to be an interestingness metric i.e the frequent sub-graph that appears in most graph databases is considered interesting



Graph Isomorphism

- FSG uses canonical labeling for isomorphism.
- Canonical labeling assigns a unique code for each substructure and two substructures have the same canonical code only if the substructures are isomorphic.
- Canonical labeling is an easier and faster way of finding the isomorphic substructures, but it suffers from the fact that canonical labeling cannot be used for graphs that have multiple edges between the vertices.



Key Aspects

- interested in subgraphs that are connected
- allow the graphs to be labeled
- both vertices and edges may have labels associated with them which are not required to be unique.

FSG

- Input to FSG
  - Set of graphs (transactions)
  - Labeled edges and vertices
  - Edges are undirected
  - No inexact match

  - Subdue can take a single connected graph or a forest of graphs
  - Edges can be directed or undirected
  - Both edges and vertices can have labels
  - Multiple edges between nodes is supported
  - Cycles are supported

Definitions

- The canonical label of a graph G = (V;E), cl(G) : unique code (e.g., string) that is invariant on the ordering of the vertices and edges in the graph.

- Two graphs will have the same canonical label if they are isomorphic.

- Canonical labels are useful to (i) compare two graphs (ii) establish a complete ordering of a set of graphs in a unique and deterministic way, regardless of the original vertex and edge ordering.

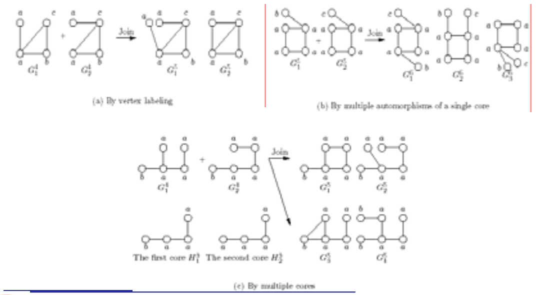
FSG

- Frequent subgraphs are found based on the set covering approach (frequency)
  - In Subdue subgraphs are found based on MDL (the graph that minimizes the description length of the input)
- User defined support threshold – minimum percentage of graphs in which a subgraph has to be found

Candidate generation

- Candidates are the substructures which would be searched and counted in the given graph databases
- create a set of candidates of size *k*+1, given frequent *k*-subgraphs.
- by joining two frequent *k*-subgraphs (using downward closure property)
- must contain the same (*k*-1)-subgraph (common *core*)
- Self-join required for unlabeled graphs
- Subdue extends subgraph in every possible way via an edge and a vertex

Joining of two k-subgraphs



(a) By vertex labeling

(b) By multiple automorphisms of a single core

The first core H_1^3 The second core H_2^3

(c) By multiple cores

Key computational steps in candidate generation

- core identification
- Joining
- using the downward closure property

Core Identification

- for each frequent *k*-subgraph, store the canonical labels of its frequent (*k*-1)-subgraphs
- Cores are the intersection of these lists.
- complexity : quadratic on |F(k)|

- *inverted indexing scheme*
- for each frequent (k-1)subgraph, maintain a list of child k-subgraphs.
- form every possible pair from the child list of every (k-1) frequent subgraph.
- complexity of finding an appropriate pair of subgraphs: square of the number of child k-subgraphs (which is much smaller)

Speeding automorphism computation

- cache previous automorphisms associated with each core
- look them up instead of performing the same automorphism computation again.
- saved list of automorphisms is discarded once *Ck*+1 has been generated.

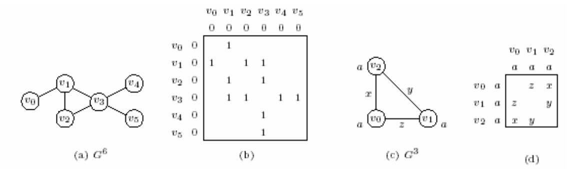
Downward Closure

- uses canonical labeling to substantially reduce the complexity of the checking whether or not a candidate pattern satisfies the downward closure property of the support condition

Canonical labels

- Canonical labels are computed for subgraphs
- These labels are used for subgraph comparison (instead of isomorphism)
- A number of optimizations are proposed to reduce the complexity from O(|V|!)
- But once computed, they can be cached and used quickly for comparison

Canonical Labeling



$$\begin{array}{cc|cccccc} & & v_0 & v_1 & v_2 & v_3 & v_4 & v_5 \\ & & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline v_0 & 0 & & 1 & & & & \\ v_1 & 0 & 1 & & 1 & 1 & & \\ v_2 & 0 & & 1 & & 1 & & \\ v_3 & 0 & & 1 & 1 & & 1 & 1 \\ v_4 & 0 & & & & 1 & & \\ v_5 & 0 & & & & 1 & & \end{array}$$

$$\begin{array}{cc|ccc} & & v_0 & v_1 & v_2 \\ & & a & a & a \\ \hline v_0 & a & & z & x \\ v_1 & a & z & & y \\ v_2 & a & x & y & \end{array}$$

"000000 1 01 011 0001 00010"

"*aaa z xy*"

- Different orderings of the vertices will give rise to different codes
- Try every possible permutation of the vertices and choose the ordering which gives lexicographically the largest, or the smallest code.
- O(|V|!)

Why canonical labeling?

- use the canonical label repeatedly for comparison without the recalculation.
- by regarding canonical labels as strings, we get the total order of graphs.
- sort them in an array
- index by binary search efficiently.

Canonical label optimizations

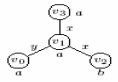
- Vertex invariants – do not change across isomorphism mappings (e.g., degree or label of a vertex)
- Do not asymptotically change the computational complexity; in practicce it is useful

Vertex Invariants

- attributes or properties assigned to a vertex which do not change across isomorphism mappings.
- partition the vertices into equivalence classes such that all the vertices assigned to the same partition have the same values for the vertex invariants.
- only maximize over those permutations that keep the vertices in each partition together.

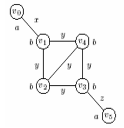
Invariants

- degree or label of a vertex
- the labels and degrees of their adjacent vertices (neighbor list)
- information about their adjacent partitions



$$\begin{array}{cc|cccc|}
 & & v_0 & v_1 & v_2 & v_3 \\
 & & a & a & b & a \\
\hline
v_0 & a & & y & & \\
v_1 & a & y & & x & x \\
v_2 & b & & x & & \\
v_3 & a & & x & & \\
\hline
\end{array}
\qquad
\begin{array}{cc|c|cc|c|}
 & & v_1 & v_0 & v_3 & v_2 \\
 & & a & a & a & b \\
\hline
v_1 & a & & x & y & x \\
\hline
v_0 & a & x & & & \\
v_3 & a & y & & & \\
\hline
v_2 & b & x & & & \\
\hline
 & & p_0 & \multicolumn{2}{c}{p_1} & p_2
\end{array}
\qquad
\begin{array}{cc|c|cc|c|}
 & & v_1 & v_3 & v_0 & v_2 \\
 & & a & a & a & b \\
\hline
v_1 & a & & y & x & x \\
\hline
v_3 & a & y & & & \\
v_0 & a & x & & & \\
\hline
v_2 & b & x & & & \\
\hline
 & & p_0 & \multicolumn{2}{c}{p_1} & p_2
\end{array}$$

only 1! * 2!= 2 permutations although the total permutations 4! = 24.

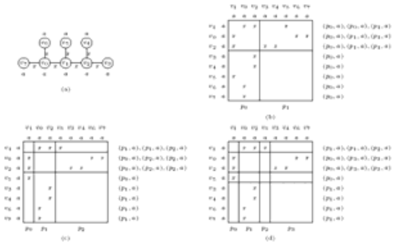


$$\begin{array}{cc|cccc|cc|}
 & & v_2 & v_4 & v_1 & v_3 & v_0 & v_5 \\
 & & b & b & b & b & a & a \\
\hline
v_2 & b & & y & y & y & & \\
v_4 & b & y & & y & y & & \\
v_1 & b & y & y & & & x & \\
v_3 & b & y & y & & & & z \\
\hline
v_0 & a & & & x & & & \\
v_5 & a & & & & z & & \\
\hline
 & & \multicolumn{4}{c}{p_0} & \multicolumn{2}{c}{p_1}
\end{array}
\qquad
\begin{array}{cc|cc|c|c|c|c|}
 & & v_2 & v_4 & v_1 & v_3 & v_0 & v_5 \\
 & & b & b & b & b & a & a \\
\hline
v_2 & b & & y & y & y & & \\
v_4 & b & y & & y & y & & \\
\hline
v_1 & b & y & y & & & x & \\
\hline
v_3 & b & y & y & & & & z \\
\hline
v_0 & a & & & x & & & \\
\hline
v_5 & a & & & & z & & \\
\hline
 & & \multicolumn{2}{c}{p_0} & p_1 & p_2 & p_3 & p_4
\end{array}$$

$(y, 3, b), (y, 3, b), (y, 3, b)$
$(y, 3, b), (y, 3, b), (y, 3, b)$
$(x, 1, a), (y, 3, b), (y, 3, b)$
$(y, 3, b), (y, 3, b), (z, 1, a)$
$(x, 3, b)$
$(z, 3, b)$

- (l(e)*;* d(v)*;* l(v)) l(e) is the label of the incident edge e, d(v) is degree of the adjacent vertex v, and l(v) is its vertex label.
- same partition if and only if nl(u) = nl(v)
- reduce from 4! * 2! to 2!.

Iterative Partitioning



Frequency Counting

- for each frequent subgraph, keep a list of transaction identifiers that support it.
- to compute the frequency of G(k+1), first compute the intersection of the TID lists of its frequent k-subgraphs.
- If the size of the intersection is below the support, G(k+1) is pruned - subgraph isomorphism computations avoided
- Otherwise use subgraph isomorphism on the set of transactions in the intersection of the TID lists.

FSG vs. Subdue

- No inexact graph matching
- No iterative discovery
- Restricts input in order to be more efficient
  - Undirected edges only
  - Set of disconnected graphs
- Optimizations rely on additional space for increased speed

gSpan

- Given a graph dataset, D = {G_0,G_1,….G_n} and any subgraph g, the problem of frequent subgraph mining is to find any subgraph g such that support(g) > minSupport
- Unlike FSG, gSpan discovers frequent substructures without candidate generation.
- gSpan builds a new lexicographic order among graphs, and maps each graph to a unique minimum DFS code as its canonical label. Based on this lexicographic order, gSpan adopts the depth-first search strategy to mine frequent connected subgraphs efficiently.
- gSpan discovers all the frequent subgraphs without candidate generation and false positives pruning. It combines the growing and checking of frequent subgraphs into one procedure, thus accelerating the mining process.

gSpan features:

- In the context of frequent subgraph mining, the Apriori-like algorithms meet two challenges:
  - candidate generation: the generation of size (k+1) subgraph candidates from size k frequent subgraphs is more complicated and costly and
  - pruning false positives: subgraph isomorphism test is an NP complete problem, thus pruning false positives is costly.

If the entire graph dataset can fit in main memory, gSpan can be applied directly; otherwise, one can first perform graph-based data projection and then apply gSpan

- Subgraph isomorphism is achieved using the canonical labeling techniques, DFS lexicographic order and minimum DFS code.

DFS Code

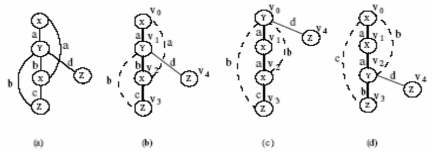
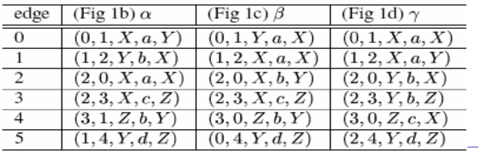


Figure 1. Depth-First Search Tree



edge	(Fig 1b) α	(Fig 1c) β	(Fig 1d) γ
0	$(0, 1, X, a, Y)$	$(0, 1, Y, a, X)$	$(0, 1, X, a, X)$
1	$(1, 2, Y, b, X)$	$(1, 2, X, a, X)$	$(1, 2, X, a, Y)$
2	$(2, 0, X, a, X)$	$(2, 0, X, b, Y)$	$(2, 0, Y, b, X)$
3	$(2, 3, X, c, Z)$	$(2, 3, X, c, Z)$	$(2, 3, Y, b, Z)$
4	$(3, 1, Z, b, Y)$	$(3, 0, Z, b, Y)$	$(3, 0, Z, c, X)$
5	$(1, 4, Y, d, Z)$	$(0, 4, Y, d, Z)$	$(2, 4, Y, d, Z)$

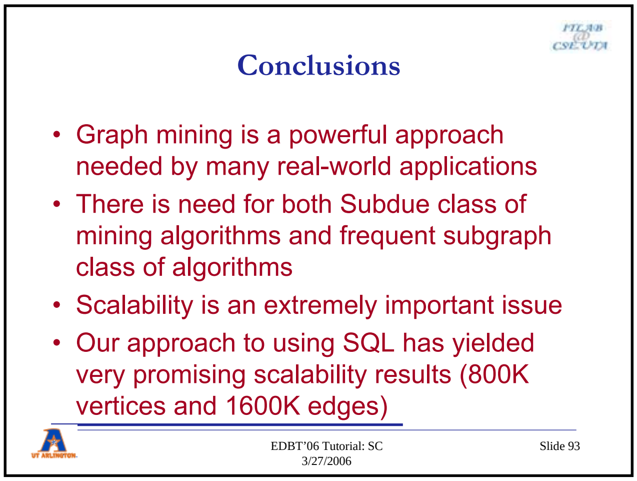
Table 1. DFS codes for Fig. 1(b)-(d)

Min DFS Code and Isomorphism

- If we order all DFS codes according to the < order, we can take a minimum, min(G)
- min(G) is unique
- Two graphs are isomorphic iff min(G) = min(G')

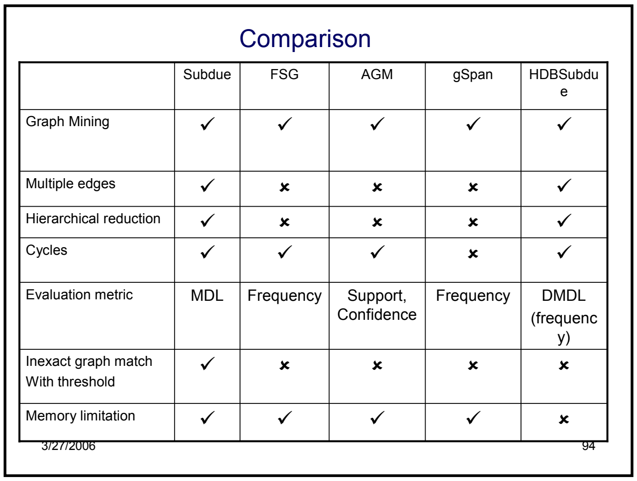
Observations

- gSpan is a main memory algorithm
- Performance is reported for data sets up to only 320 KB
- Running time scales exponentially with large numbers of graph labels
- gSpan typically needs random access to elements of the graph database and to its projections



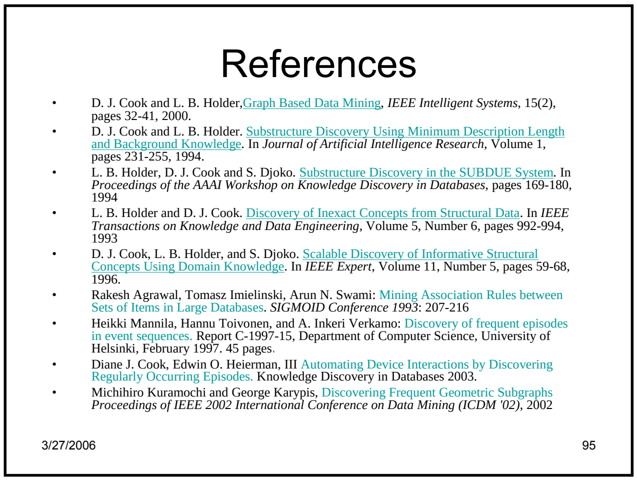
Conclusions

- Graph mining is a powerful approach needed by many real-world applications
- There is need for both Subdue class of mining algorithms and frequent subgraph class of algorithms
- Scalability is an extremely important issue
- Our approach to using SQL has yielded very promising scalability results (800K vertices and 1600K edges)



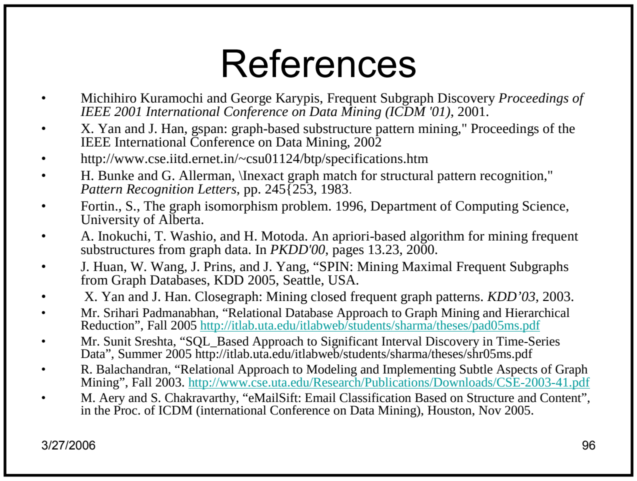
Comparison

	Subdue	FSG	AGM	gSpan	HDBSubdue
Graph Mining	✓	✓	✓	✓	✓
Multiple edges	✓	✗	✗	✗	✓
Hierarchical reduction	✓	✗	✗	✗	✓
Cycles	✓	✓	✓	✗	✓
Evaluation metric	MDL	Frequency	Support, Confidence	Frequency	DMDL (frequency)
Inexact graph match With threshold	✓	✗	✗	✗	✗
Memory limitation	✓	✓	✓	✓	✗



References

- D. J. Cook and L. B. Holder,Graph Based Data Mining, *IEEE Intelligent Systems*, 15(2), pages 32-41, 2000.
- D. J. Cook and L. B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. In *Journal of Artificial Intelligence Research*, Volume 1, pages 231-255, 1994.
- L. B. Holder, D. J. Cook and S. Djoko. Substructure Discovery in the SUBDUE System. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pages 169-180, 1994
- L. B. Holder and D. J. Cook. Discovery of Inexact Concepts from Structural Data. In *IEEE Transactions on Knowledge and Data Engineering*, Volume 5, Number 6, pages 992-994, 1993
- D. J. Cook, L. B. Holder, and S. Djoko. Scalable Discovery of Informative Structural Concepts Using Domain Knowledge. In *IEEE Expert*, Volume 11, Number 5, pages 59-68, 1996.
- Rakesh Agrawal, Tomasz Imielinski, Arun N. Swami: Mining Association Rules between Sets of Items in Large Databases. *SIGMOID Conference 1993*: 207-216
- Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo: Discovery of frequent episodes in event sequences. Report C-1997-15, Department of Computer Science, University of Helsinki, February 1997. 45 pages.
- Diane J. Cook, Edwin O. Heierman, III Automating Device Interactions by Discovering Regularly Occurring Episodes. Knowledge Discovery in Databases 2003.
- Michihiro Kuramochi and George Karypis, Discovering Frequent Geometric Subgraphs *Proceedings of IEEE 2002 International Conference on Data Mining (ICDM '02)*, 2002



References

- Michihiro Kuramochi and George Karypis, Frequent Subgraph Discovery *Proceedings of IEEE 2001 International Conference on Data Mining (ICDM '01)*, 2001.
- X. Yan and J. Han, gspan: graph-based substructure pattern mining," Proceedings of the IEEE International Conference on Data Mining, 2002
- http://www.cse.iitd.ernet.in/~csu01124/btp/specifications.htm
- H. Bunke and G. Allerman, \Inexact graph match for structural pattern recognition," *Pattern Recognition Letters*, pp. 245{253, 1983.
- Fortin., S., The graph isomorphism problem. 1996, Department of Computing Science, University of Alberta.
- A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD'00*, pages 13.23, 2000.
- J. Huan, W. Wang, J. Prins, and J. Yang, "SPIN: Mining Maximal Frequent Subgraphs from Graph Databases, KDD 2005, Seattle, USA.
- X. Yan and J. Han. Closegraph: Mining closed frequent graph patterns. *KDD'03*, 2003.
- Mr. Srihari Padmanabhan, "Relational Database Approach to Graph Mining and Hierarchical Reduction", Fall 2005 http://itlab.uta.edu/itlabweb/students/sharma/theses/pad05ms.pdf
- Mr. Sunit Sreshta, "SQL_Based Approach to Significant Interval Discovery in Time-Series Data", Summer 2005 http://itlab.uta.edu/itlabweb/students/sharma/theses/shr05ms.pdf
- R. Balachandran, "Relational Approach to Modeling and Implementing Subtle Aspects of Graph Mining", Fall 2003. http://www.cse.uta.edu/Research/Publications/Downloads/CSE-2003-41.pdf
- M. Aery and S. Chakravarthy, "eMailSift: Email Classification Based on Structure and Content", in the Proc. of ICDM (international Conference on Data Mining), Houston, Nov 2005.