

Graph Mining

Hassan Sayyadi
sayyadi@cs.umd.edu

Department of computer Science
University of Maryland-College Park

Shanchan Wu
wsc@cs.umd.edu

Department of computer Science
University of Maryland-College Park



Outline

- **Introduction**
- **The Apriori for subGraph Mining (fsg)**
- **gSpan: Graph-Based Substructure Pattern Mining**
- **Some Other Mining Methods**
- **Q and A**





Frequent Pattern Extraction

- **Traditional Data**
 - Association Rules
 - Market Basket

- **Graph and Networks**
 - Frequent Subgraphs





Previously on Graph Mining

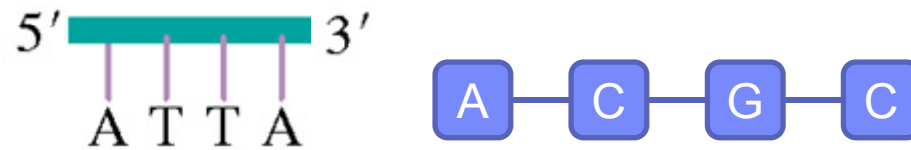
- **Network cascades: observations, models and algorithms**
 - Cascading Behavior in Large Blog Graphs Patterns and a model
- **Information Diffusion Survey**
 - Patterns of Influence in a Recommendation Network



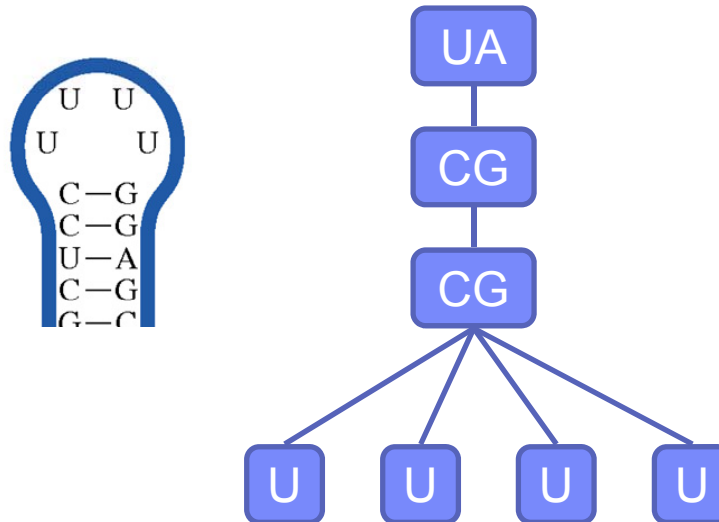


Other Applications

- **DNA Sequence**

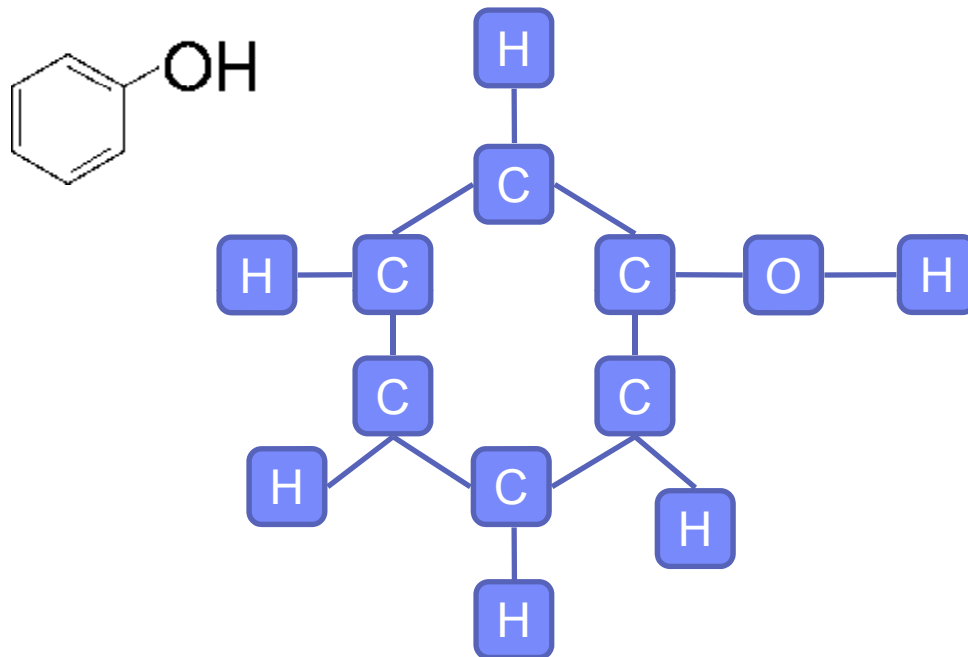


- **RNA**



Other Applications

- **Compounds**



- **Texts in literature**

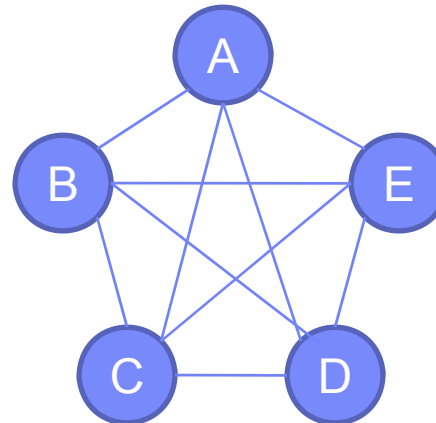




Association Rule Mining

- **Market Basket**
- **Special Case of sub-Graph Mining**
 - Each transaction is a complete graph

- **{A,B,C,D,E}**





The model: data

- **$I = \{i_1, i_2, \dots, i_m\}$: a set of items.**
- **Transaction t :**
 - t a set of items, and $t \subseteq I$.
- **Transaction Database T : a set of transactions $T = \{t_1, t_2, \dots, t_n\}$.**





The Apriori algorithm

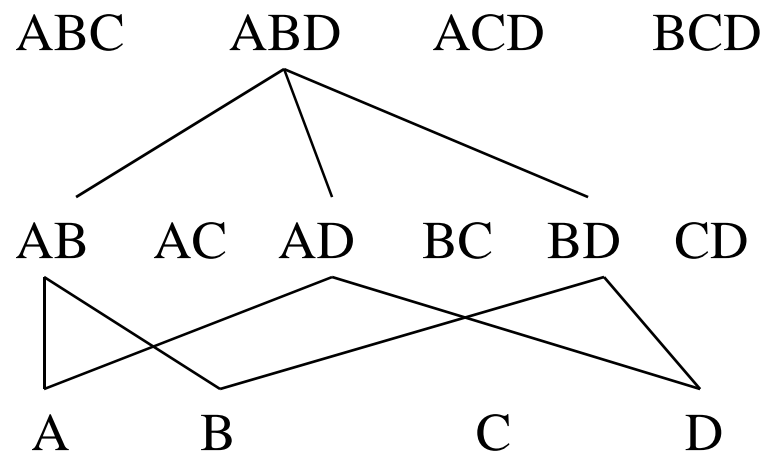
- **Probably the best known algorithm**
- **Two steps:**
 - Find all itemsets that have minimum support
 - Use frequent itemsets to generate rules.
- **E.g., a frequent itemset**
 - {Chicken, Clothes, Milk} [sup = 3/7]
- **and one rule from the frequent itemset**
 - Clothes \rightarrow Milk, Chicken [sup = 3/7, conf = 3/3]





Step 1: Mining all frequent itemsets

- A frequent itemset is an itemset whose support is $\geq \text{minsup}$.
- Key idea: The apriori property
 - any subsets of a frequent itemset are also frequent itemsets.





The Algorithm

- **Iterative algo.:** Find all 1-item frequent itemsets; then all 2-item frequent itemsets, and so on.
 - In each iteration k , only consider itemsets that contain some $k-1$ frequent itemset.
 - C_k = candidates of size k : those itemsets of size k that could be frequent, given F_{k-1}
 - F_k = those itemsets that are actually frequent, $F_k \subseteq C_k$ (need to scan the database once).





Graph Data

- **Graph isomorphism**
- **Graph representation**
- **Canonical labeling**





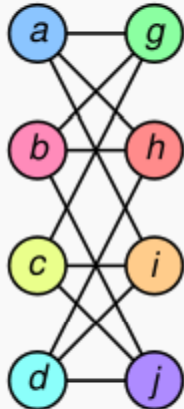
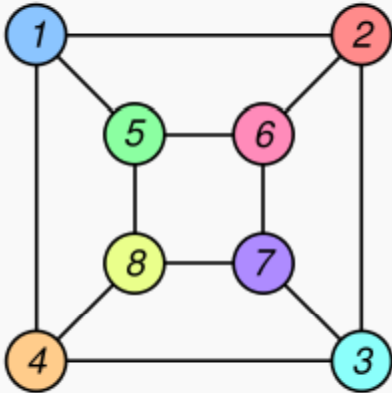
Isomorphism

- **Graph Isomorphism**
 - The Problem of determining whether given two graph are isomorphic.
- **Authomorphism**
- **Subgraph Isomorphism**
- **P or NP-complete?**



Isomorphism

- **isomorphism of graphs G and H is a bijection between the vertex sets of G and H**
 - $F: V(G) \rightarrow V(H)$
 - such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H .

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$



Graph Representation

■ Canonical Labeling

— Flattened representation

- M. Kuramochi and G. Karypis. Frequent subgraph discovery. In ICDM'01, pages 313–320, Nov. 2001.

— DFS code

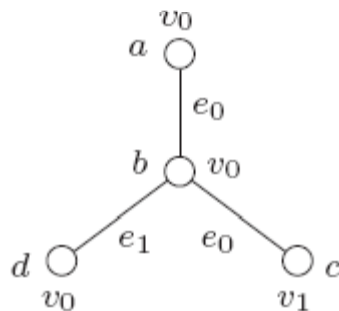
- Yan, X. and Han, J. 2002. gSpan: Graph-Based Substructure Pattern Mining. In Proceedings of the 2002 IEEE international Conference on Data Mining (Icdm'02) (December 09 - 12, 2002). ICDM. IEEE Computer Society, Washington, DC, 721.





Flattened Representation

1



2

id	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
label	<i>v</i> ₀	<i>v</i> ₀	<i>v</i> ₁	<i>v</i> ₀
<i>a</i>	0	<i>e</i> ₀	0	0
<i>b</i>	<i>e</i> ₀	0	<i>e</i> ₀	<i>e</i> ₁
<i>c</i>	0	<i>e</i> ₀	0	0
<i>d</i>	0	<i>e</i> ₁	0	0

3

id	<i>a</i>	<i>c</i>	<i>d</i>	<i>b</i>
label	<i>v</i> ₀	<i>v</i> ₁	<i>v</i> ₀	<i>v</i> ₀
partition	0			1
<i>a</i>	0	0	0	<i>e</i> ₀
<i>c</i>	0	0	0	<i>e</i> ₀
<i>d</i>	0	0	0	<i>e</i> ₁
<i>b</i>	<i>e</i> ₀	<i>e</i> ₀	<i>e</i> ₁	0

4

id	<i>d</i>	<i>a</i>	<i>c</i>	<i>b</i>
label	<i>v</i> ₀	<i>v</i> ₀	<i>v</i> ₁	<i>v</i> ₀
partition	0		1	2
<i>d</i>	0	0	0	<i>e</i> ₁
<i>a</i>	0	0	0	<i>e</i> ₀
<i>c</i>	0	0	0	<i>e</i> ₀
<i>b</i>	<i>e</i> ₁	<i>e</i> ₀	<i>e</i> ₀	0





Flattened Representation

id	<i>d</i>	<i>a</i>	<i>c</i>	<i>b</i>
label	v_0	v_0	v_1	v_0
partition	0		1	2
<i>d</i>	0	0	0	e_1
<i>a</i>	0	0	0	e_0
<i>c</i>	0	0	0	e_0
<i>b</i>	e_0	e_1	e_0	0

id	<i>a</i>	<i>d</i>	<i>c</i>	<i>b</i>
label	v_0	v_0	v_1	v_0
partition	0		1	2
<i>a</i>	0	0	0	e_0
<i>d</i>	0	0	0	e_1
<i>c</i>	0	0	0	e_0
<i>b</i>	e_0	e_1	e_0	0

000e1e0e0

>

000e0e1e0

- **2 VS 4!=24 permutations**





Isomorphism

- **Select a single node from G_1**
- **Find a mapping to one node from G_2**
- **Add vertices one by one until**
 - Finding a complete mapping
 - Exhausting the search space





The Apriori for subGraph Mining

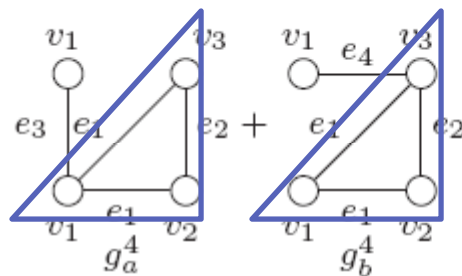
■ **Iterative algo.:**

- Find all frequent 1-subGraphs;
- In each iteration k , only consider frequent $(k-1)$ -subgraphs
 - C_k = candidates of size k : those subgraphs of size k that could be frequent, given F_{k-1}
 - F_k = those subgraphs that are actually frequent, $F_k \subseteq C_k$ (need to scan the database once).



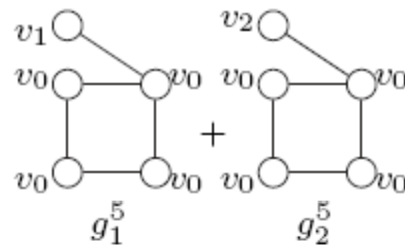
Candidate Generation

- **Create a set of candidate size $k+1$**
 - from given two frequent k -subgraphs
 - Containing the same $(k-1)$ -subgraph
 - Result in several candidates size $k+1$





Candidate Generation





Frequency Counting

- **Transaction ID list**
 - For each frequent subgraph
- **Frequency of $(k+1)$ -subgraph**
 - If the size of the intersection of TID lists is below the support
 - Prune
 - Otherwise
 - Compute the frequency by graph isomorphism





Experiments

N	I	T	Running Time [sec]	
			$\sigma = 2\%$	$\sigma = 1\%$
2	3	5	18	24
		10	143	434
		20	—	—
		40	—	—
2	5	5	27	52
		10	251	2246
		20	—	—
		40	—	—
2	7	10	557	6203
		20	—	—
		40	—	—
2	10	10	—	—
		20	—	—

N	I	T	Running Time [sec]	
			$\sigma = 2\%$	$\sigma = 1\%$
3	3	5	12	22
		10	30	40
		20	112	390
		40	5817	—
3	5	5	18	32
		10	51	102
		20	189	736
		40	6110	—
3	7	10	66	4512
		20	1953	—
		40	—	—
3	10	10	8290	—
		20	—	—

N	I	T	Running Time [sec]	
			$\sigma = 2\%$	$\sigma = 1\%$
5	3	5	10	12
		10	20	25
		20	53	71
		40	196	279
5	5	5	24	44
		10	55	80
		20	124	174
		40	340	617
5	7	10	208	770
		20	772	1333
		40	2531	3143
5	10	10	10914	—
		20	—	—

N	I	T	Running Time [sec]	
			$\sigma = 2\%$	$\sigma = 1\%$
10	3	5	9	17
		10	16	25
		20	35	40
		40	87	98
10	5	5	10	18
		10	20	51
		20	47	119
		40	188	246
10	7	10	190	816
		20	866	1506
		40	2456	3199
10	10	10	10785	—
		20	—	—

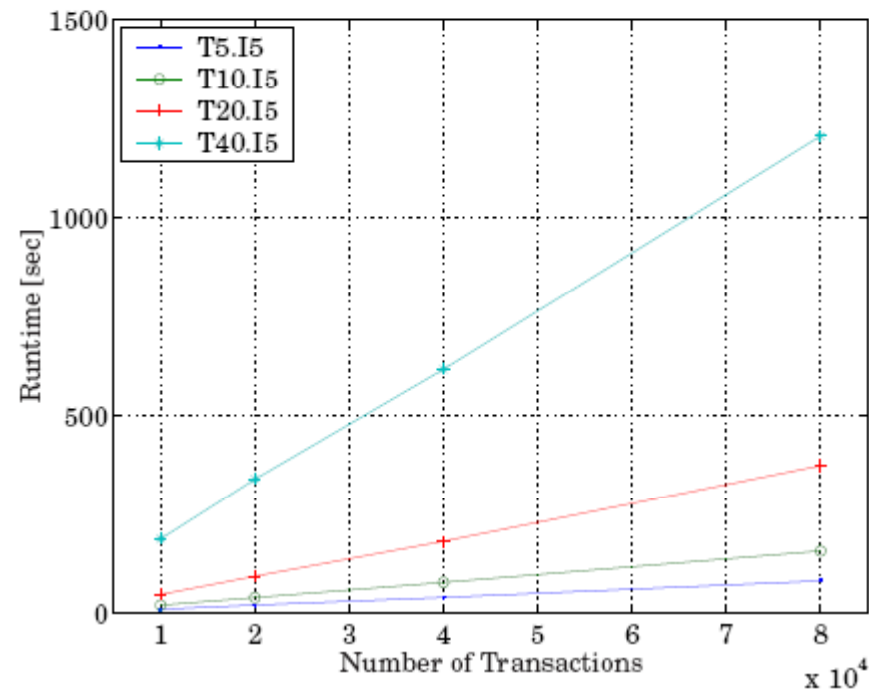
N	I	T	Running Time [sec]	
			$\sigma = 2\%$	$\sigma = 1\%$
20	3	5	9	16
		10	16	28
		20	34	38
		40	78	85
20	5	5	10	19
		10	20	51
		20	48	117
		40	182	233
20	7	10	193	804
		20	884	1667
		40	2524	3271
20	10	10	10520	—
		20	—	—

N	I	T	Running Time [sec]	
			$\sigma = 2\%$	$\sigma = 1\%$
40	3	5	20	22
		10	27	44
		20	44	47
		40	84	89
40	5	5	20	28
		10	29	60
		20	55	131
		40	177	234
40	7	10	197	1236
		20	861	5273
		40	2456	9183
40	10	10	9687	—
		20	—	—



Experiments

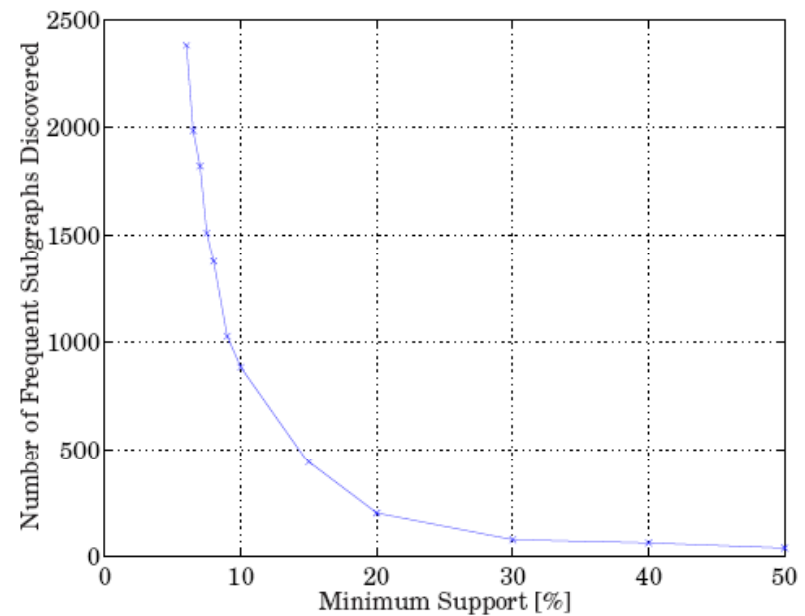
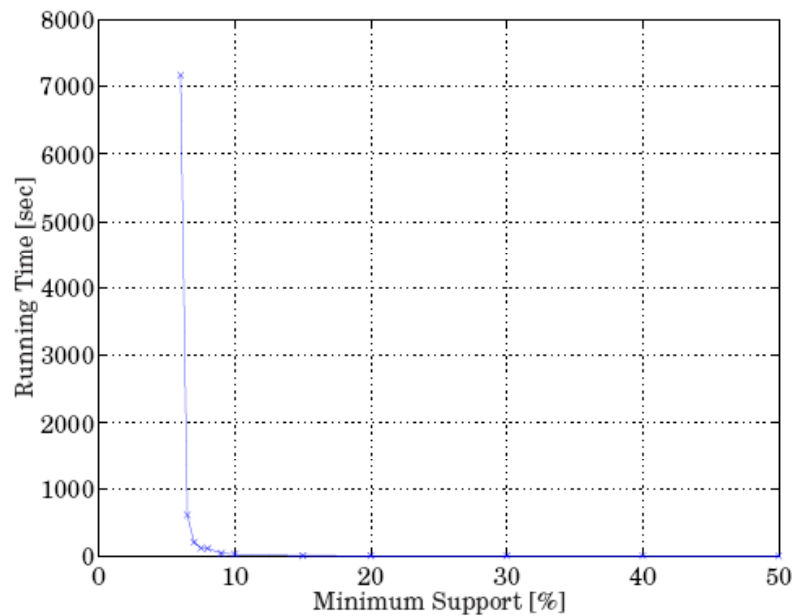
- **Synthetic data**





Experiments

■ Chemical Compound Dataset





Frequent Disconnected Graphs

- **Find all frequent connected subgraphs**
- **Convert the problem to the market basket problem**
 - Each frequent connected component is an item
- **Find all frequent itemsets**





gSpan: Graph-Based Substructure Pattern Mining

Xifeng Yan, Jiawei Han



Challenges for Apriori-like Algorithm

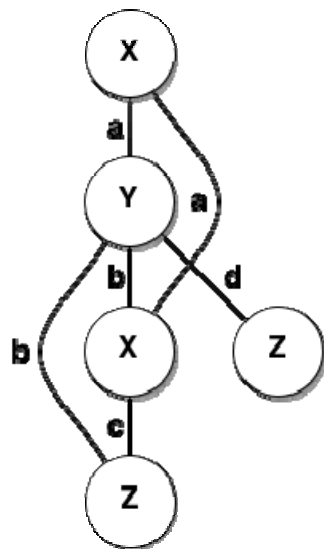
- **Generation of subgraph candidates is complicated and expensive.**
- **Subgraph isomorphism is an NP-complete problem, so pruning is expensive.**



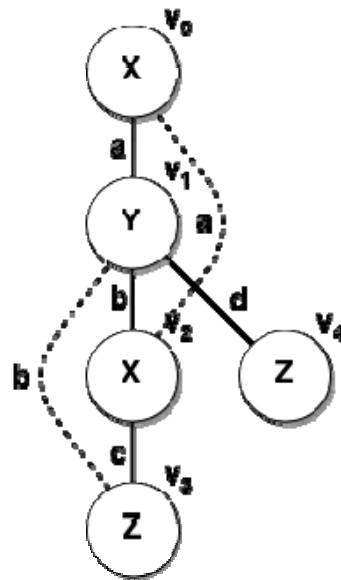
gSpan Overview

- Relabels graph representation to support DFS
- Discovers frequent substructures without candidate generation.
- Builds a new lexicographic ordering among graphs and maps each graph to a unique minimum DFS code as its canonical label. Based on this lexicographic order, gSpan enumerates not only all frequent subgraphs but also induced subgraphs.

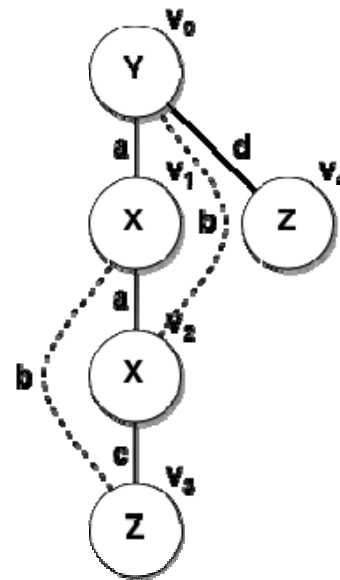
Depth-First Search Tree



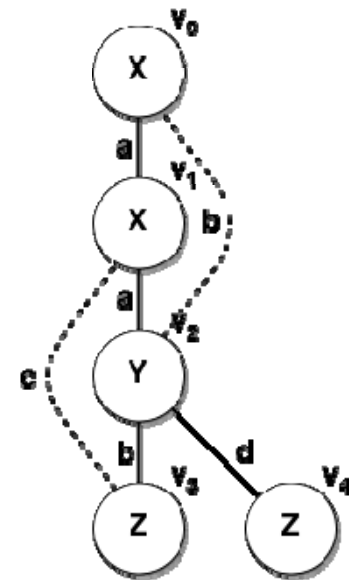
(a)



(b)



(c)



(d)

DFS Codes

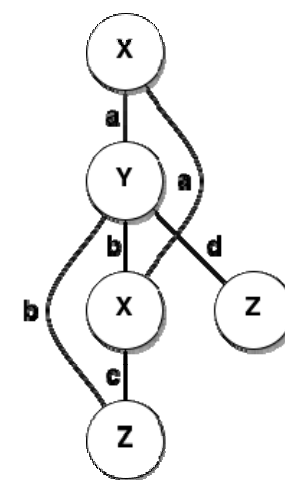
edge (i,j) is **Forward Edge** if $i < j$;
 edge (i,j) is **Backward Edge** if $i > j$

* Given $e_i = (i_1, j_1)$, $e_2 = (i_2, j_2)$: $e_1 <_T e_2$ if:
 $i_1 = i_2 \ \&\& \ j_1 < j_2$
 or $i_1 < j_1 \ \&\& \ j_1 = i_2$

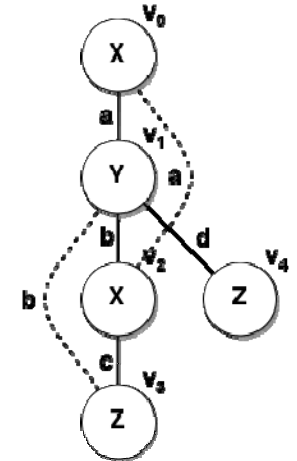
* if $e_1 <_T e_2$ and $e_2 <_T e_3$, then $e_1 <_T e_3$

code(G,T) = edge sequence of $e_i <_T e_{i+1}$

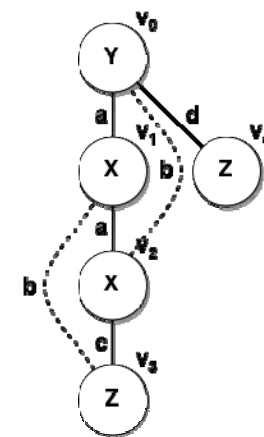
edge	(b)	(c)	(d)
0	(0,1,X,a,Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	(1,2,X,a,X)	(1,2,X,a,Y)
2	(2,0,X,a,X)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	(3,0,Z,b,Y)	(3,0,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)



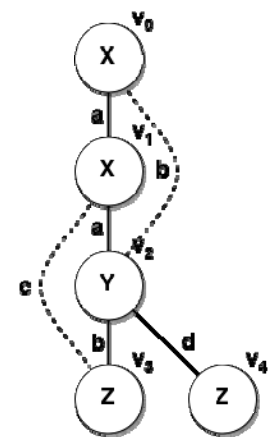
(a)



(b)



(c)



(d)



DFS Codes Construction

- First, add a new vertex and a forward edge that connects one vertex in the old code with this new vertex
- Then, add all backward edges that connect this new vertices in the old code.
- Repeat this procedure to grow the code until all edges are included in the DFS code.

DFS Lexicographic Order

- $\partial = \text{code}(\mathbf{G}_\partial, \mathbf{T}_\partial) = (a_0, a_1, \dots, a_m)$
- $\beta = \text{code}(\mathbf{G}_\beta, \mathbf{T}_\beta) = (b_0, b_1, \dots, b_n)$
- $\partial \leq \beta$ iff (1) or (2):
- (1) $\exists t, 0 \leq t \leq \min(m, n), a_k = b_k$ for $k < t, a_t <_e b_t$
- (2) $a_k = b_k$ for $0 \leq k \leq m, n \geq m$
- **Minimum DFS code**
 - The minimum DFS code $\min(G)$, in DFS lexicographic order, is the canonical label of graph G .
 - Graphs A and B are isomorphic if $\min(A) = \min(B)$.

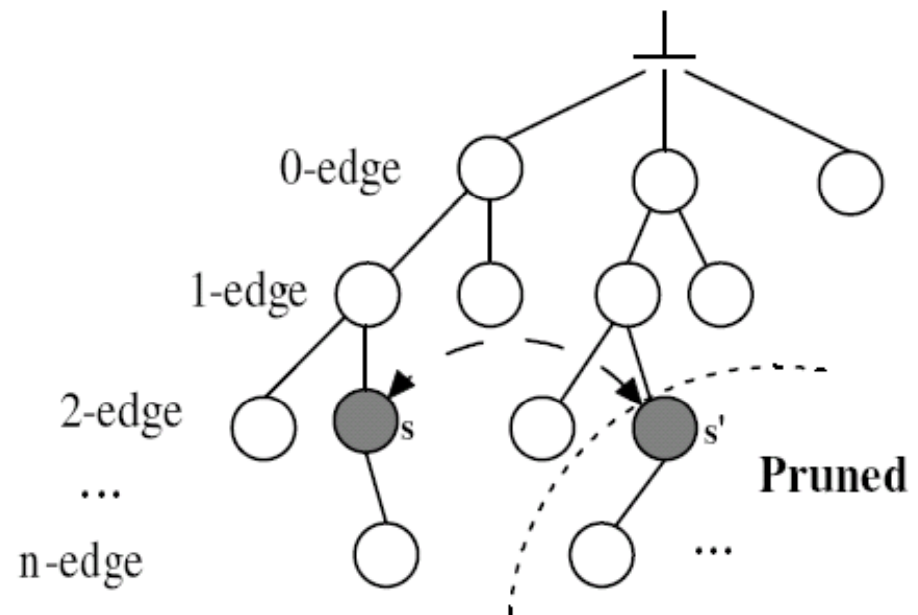


DFS Codes: Parents and Children

- If $\partial = (a_0, a_1, \dots, a_m)$ and $\beta = (a_0, a_1, \dots, a_m, b)$:
 - β is the child of ∂ .
 - ∂ is the parent of β .
- A valid DFS code requires that b grows from a vertex on the **rightmost path**.

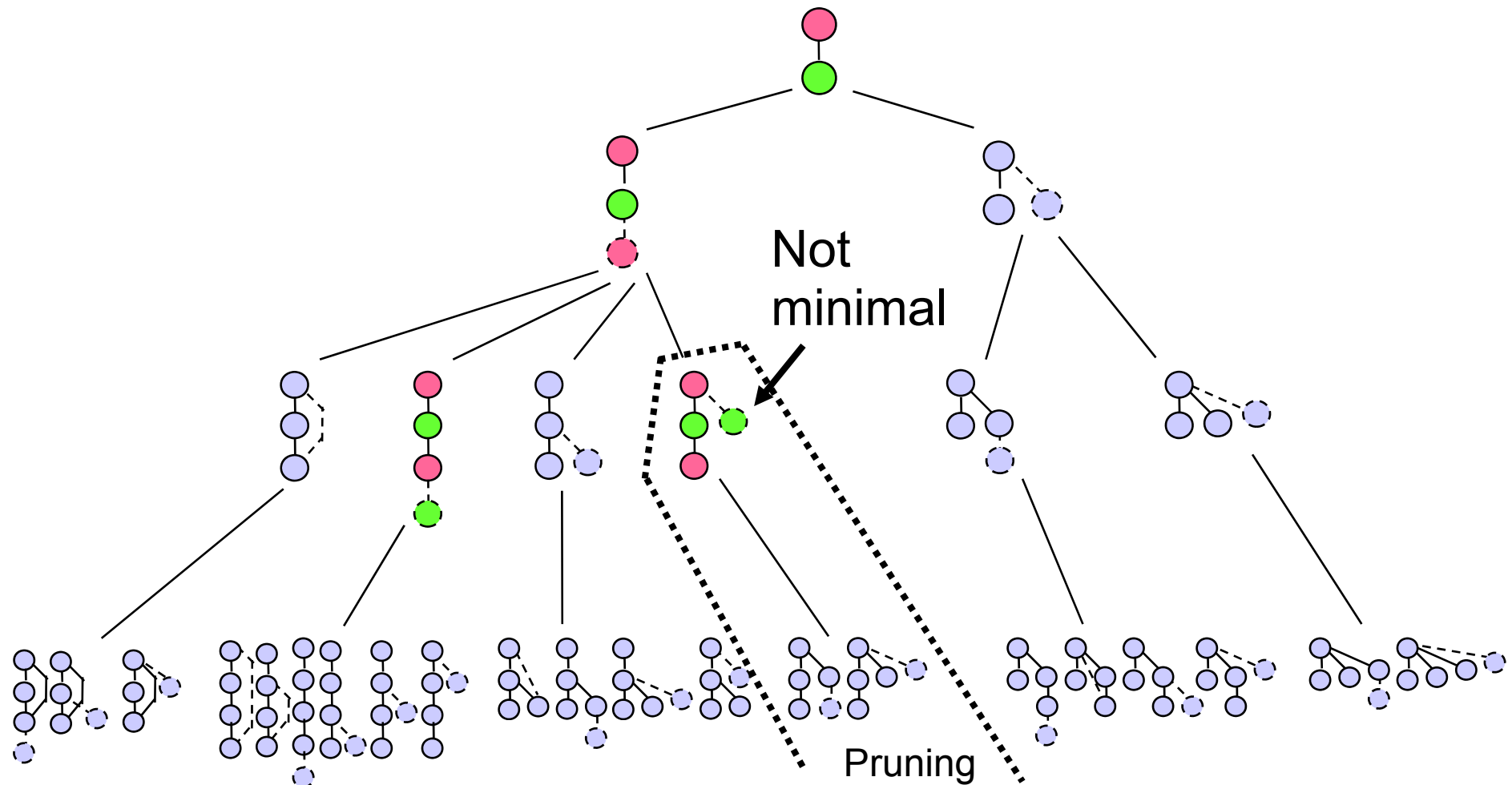
DFS Code Trees

- Organize DFS code nodes as parent-child relation.
- The relation among siblings is consistent with the DFS lexicographic order.
- The pre-order search of DFS Code Tree follows the DFS lexicographic order.
- If s and s' are the same graph with different DFS codes, s' is not the minimum and can be pruned.





DFS Code Trees - Pruning



The gSpan Algorithm

Algorithm 1 GraphSet_Projection(\mathbb{D}, \mathbb{S}).

```
1: sort the labels in  $\mathbb{D}$  by their frequency;
2: remove infrequent vertices and edges;
3: relabel the remaining vertices and edges;
4:  $\mathbb{S}^1 \leftarrow$  all frequent 1-edge graphs in  $\mathbb{D}$ ;
5: sort  $\mathbb{S}^1$  in DFS lexicographic order;
6:  $\mathbb{S} \leftarrow \mathbb{S}^1$ ;
7: for each edge  $e \in \mathbb{S}^1$  do
8:   initialize  $s$  with  $e$ , set  $s.D$  by graphs which contains  $e$ ;
9:   Subgraph_Mining( $\mathbb{D}, \mathbb{S}, s$ );
10:   $\mathbb{D} \leftarrow \mathbb{D} - e$ ;
11:  if  $|\mathbb{D}| < minSup$ ;
12:    break;
```

Subprocedure 1 Subgraph_Mining($\mathbb{D}, \mathbb{S}, s$).

```
1: if  $s \neq min(s)$ 
2:   return;
3:  $\mathbb{S} \leftarrow \mathbb{S} \cup \{s\}$ ;
4: enumerate  $s$  in each graph in  $\mathbb{D}$  and count its children;
5: for each  $c$ ,  $c$  is  $s$ ' child do
6:   if  $support(c) \geq minSup$ 
7:      $s \leftarrow c$ ;
8:     Subgraph_Mining( $\mathbb{D}_s, \mathbb{S}, s$ );
```



Algorithm Analysis

- **No candidate generation and false test.**
 - Frequent $(k+1)$ -edge subgraphs grow from k -edge frequent subgraphs directly.
- **Space saving from Dept-First Search.**
 - Apriori-like ones such as FSG adopt breadth first search strategy.
- **Quickly shrunk graph dataset**
 - At each iteration the mining procedure is performed in such a way that the whole graph dataset is shrunk to the one containing a smaller set of graphs, with each having less edges and vertices.



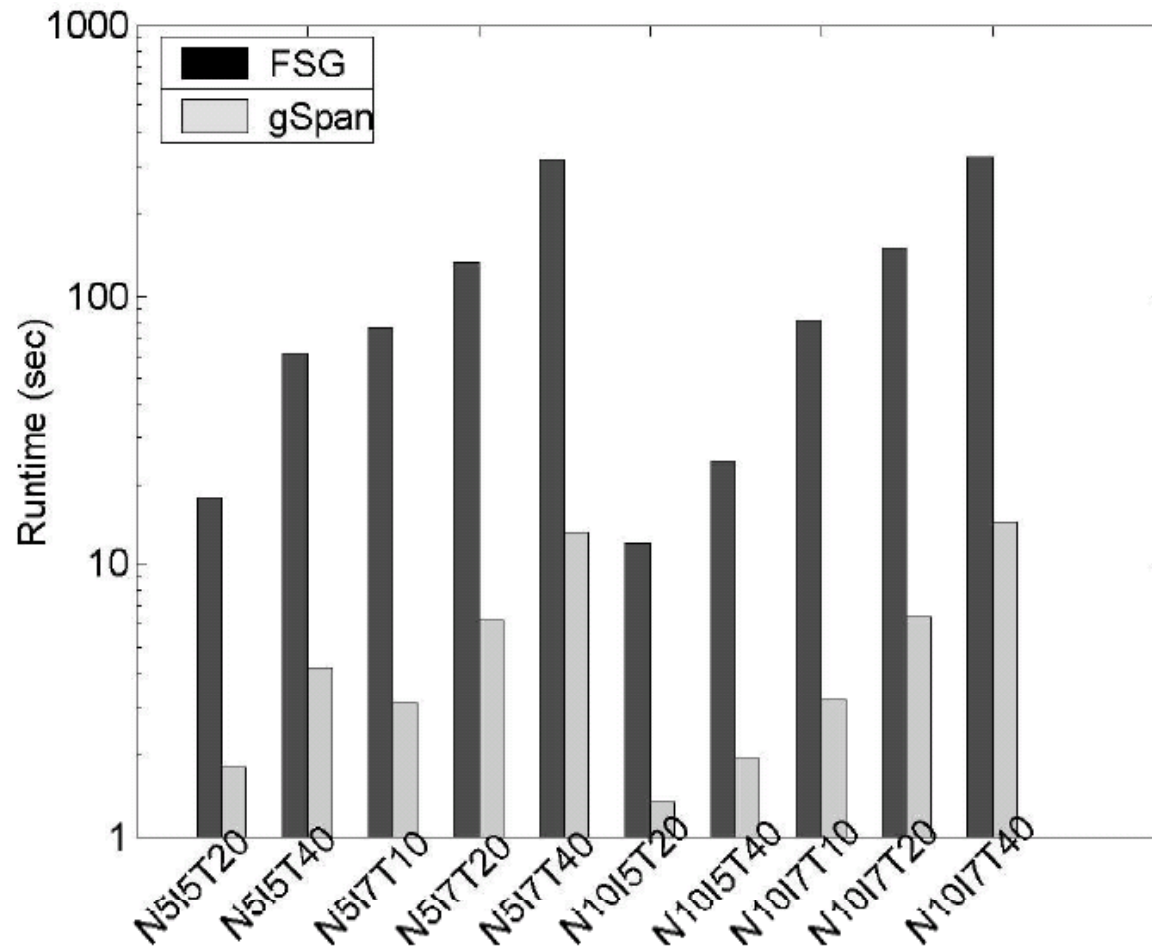
Bounded Running Time

- **The running time can be bounded by $O(kFS + rF)$**
 - Measured by the number of subgraph and/or graph isomorphism tests.
 - k is the maximum number of subgraph isomorphisms existing between a frequent subgraph and a graph in the dataset
 - F is the number of frequent subgraphs
 - S is the dataset size
 - r is the maximum number of duplicate codes of a frequent subgraph that grow from other minimum codes.

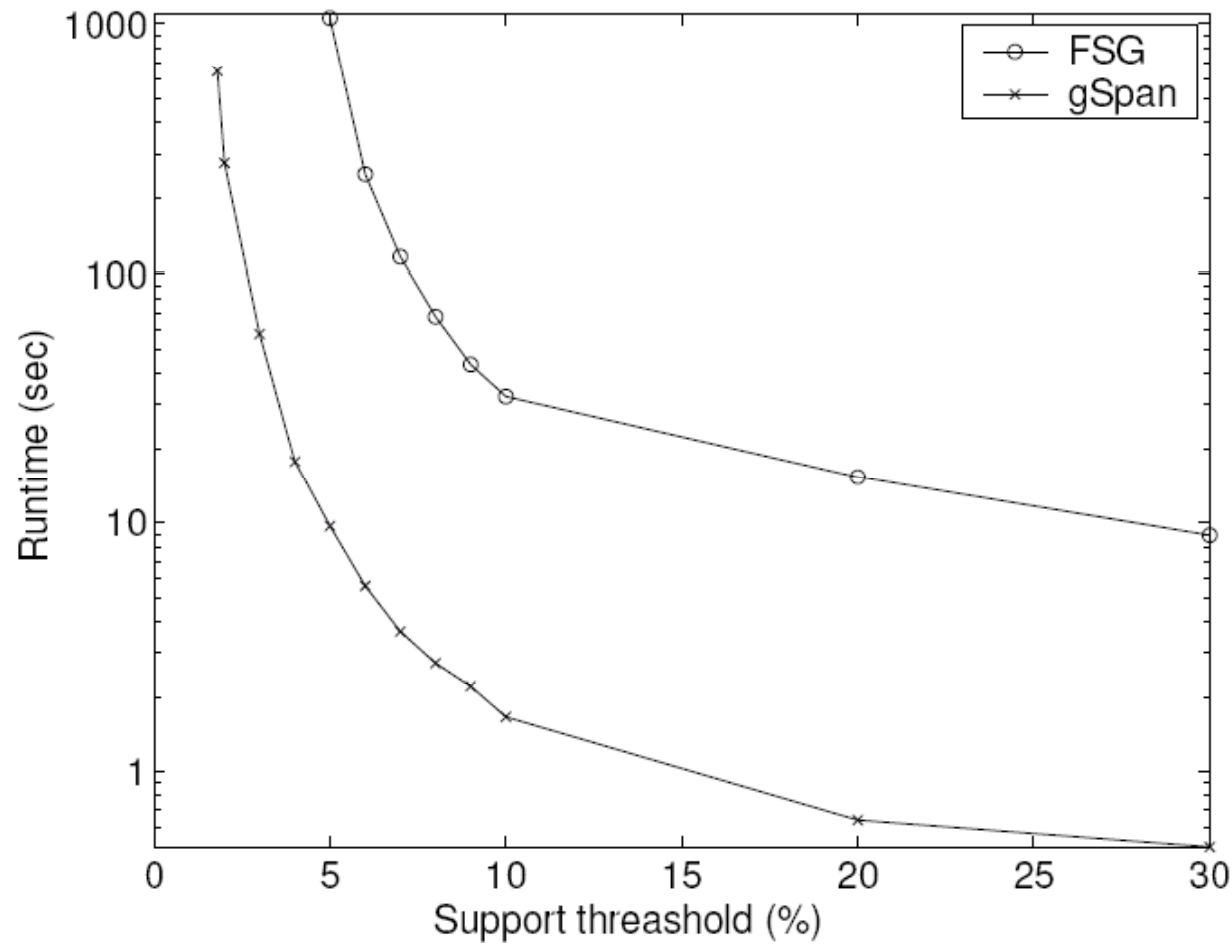
For more detail, refer to the technical report: [UIUCDCS-R-2002-2296](#)



Runtime: Synthetic data



Runtime: Chemical data





Some Other Mining Methods

- **Subdue: Compression-Based Frequent Pattern Discovery in Graph Data, OSDM05**
 - By Nikhil S. Ketkar, Lawrence B. Holder, fdsDiane J. Cook
- **CloseGraph: Mining Closed Frequent Graph Patterns, KDD'03**
 - By X. Yan and J. Han





Subdue System

- **Input: A graph G**
- **Output: A list of substructures that compress G well**





Heuristic of Subdue

- **SUBDUE uses the Minimum Description Length Principle to evaluate substructures.**
- **Description Length of a graph is the number of bits needed to send the graph's adjacency matrix to a remote computer.**
- **Goal is to minimize $DL(S) + DL(G|S)$.**



Minimum Description Length Encoding of Graphs

- Determine the number of bits **vbits** needed to encode the vertex labels of the graph
 - $\text{vbits} = \lg v + v \lg l_u$
- Determine the number of bits **rbits** needed to encode the rows of the adjacency matrix
- Determine the number of bits **ebits** needed to encode the edges represented by the entries $A[i; j] = 1$ of the adjacency matrix A .
- The total encoding of the graph takes **(vbits + rbits + ebits)** bits





Subdue Procedure

- The substructure discovery algorithm used by Subdue is a computationally-constrained beam search.
- The algorithm begins with the substructure matching a single vertex in the graph. Each iteration through the algorithm selects the best substructure and expands the instances of the substructure by one neighboring edge in all possible ways.
- The new unique generated substructures become candidates for further expansion. The algorithm searches for the best substructure until all possible substructures have been considered or the total amount of computation exceeds a given limit.
- The evaluation of each substructure is guided by the MDL principle and other background knowledge provided by the user.



Subdue Parameters

- Iterations: Graph is compressed using the best substructure, discovery is restarted
- Threshold: Controls how much two subgraphs can differ to be considered similar
- Beam Width: The number of substructures in the expansion list





Hierarchical Concept Discovery

- After a substructure is discovered, each instance of the substructure in the input graph can be replaced by a single vertex representing the entire substructure.
- The discovery procedure can then be repeated on the compressed data set, resulting in new interesting substructures.
- If the newly-discovered substructures are defined in terms of existing substructure concepts, the substructure definitions form a hierarchy of substructure concepts.



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**



Thank you

Q and A

