Mining Frequent Subgraphs

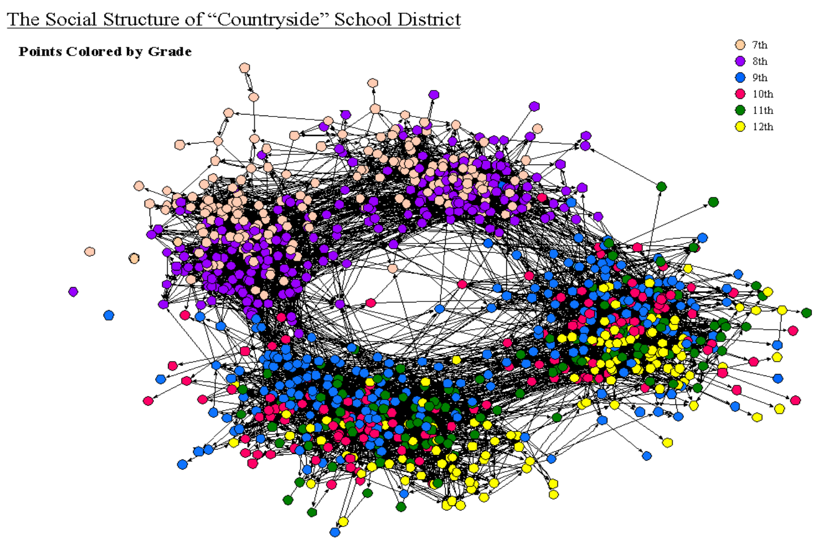
CS 145

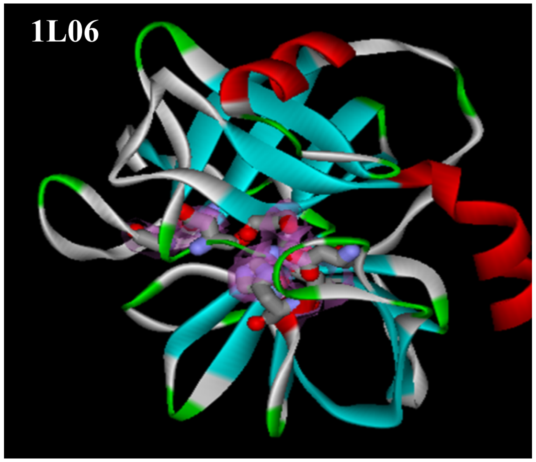
Fall 2015

The UNIVERSITY *of* CALIFORNIA *at* LOS ANGELES

Overview

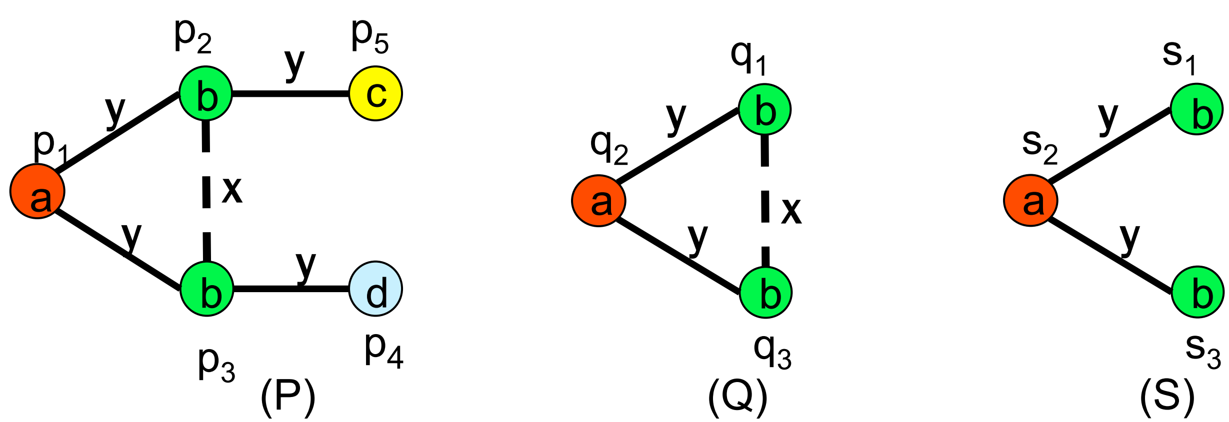
- Introduction
  - Finding recurring subgraphs from graph databases.
  - FSG
  - gSpan
  - FFSM





Labeled Graph

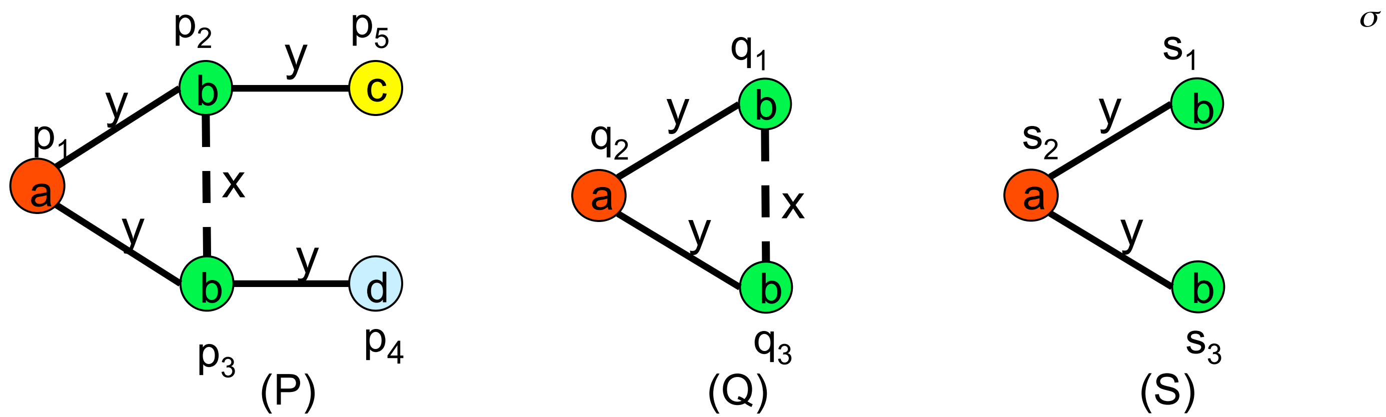
- We define a *labeled graph* G as a five element tuple $G = \{V, E, \sum_V, \sum_E, \delta\}$ where
  - V is the set of vertices of G,
  - $E \subseteq V \times V$ is a set of undirected edges of G,
  - $\sum_V$ ($\sum_E$) are set of vertex (edge) labels,
  - δ is the labeling function: $V \rightarrow \sum_V$ and $E \rightarrow \sum_E$ that maps vertices and edges to their labels.



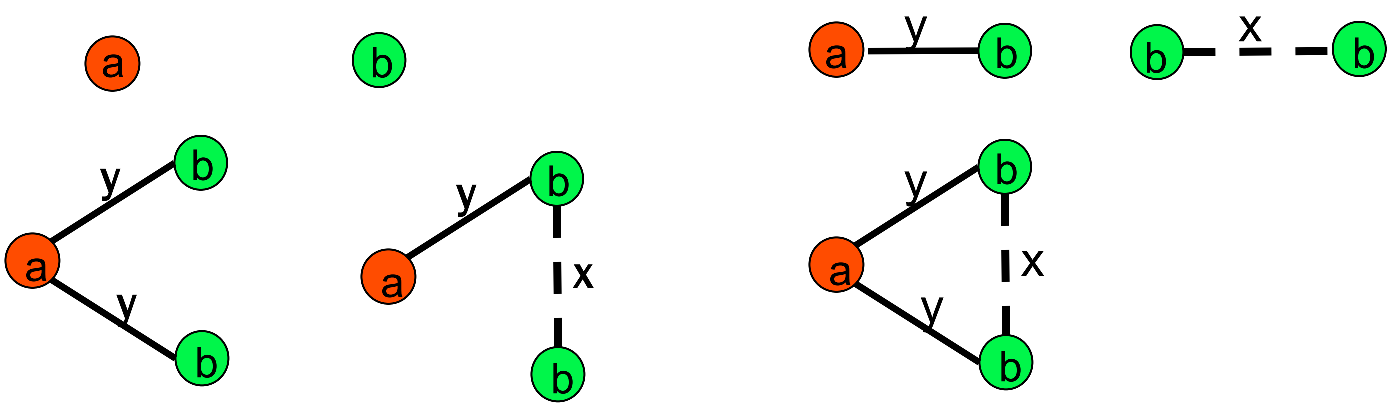
Frequent Subgraph Mining

Input: A set *GD* of labeled undirected graphs

σ = 2/3



Output: All frequent subgraphs (w. r. t. σ) from *GD*.

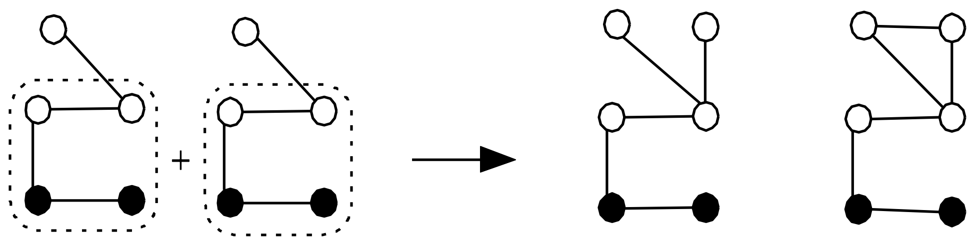


Finding Frequent Subgraphs

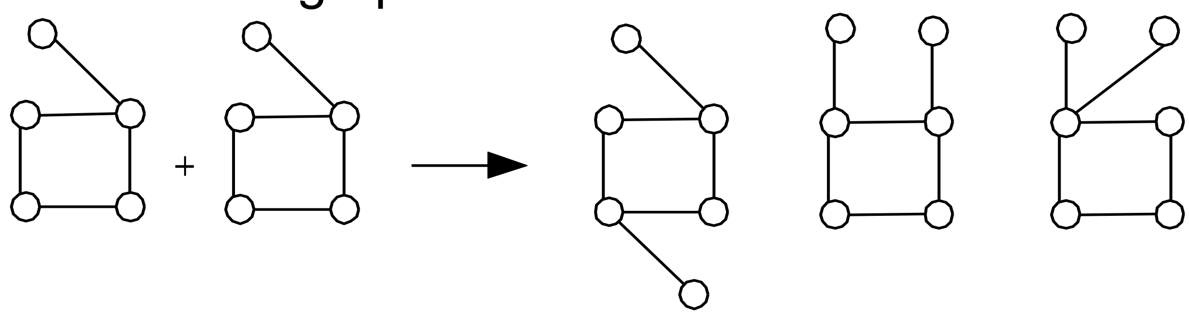
- Given a graph database GD = $\{G_0, G_1, \ldots, G_n\}$, find all subgraphs appearing in at least σ graphs.
  - Isomorphic subgraphs are considered the same subgraph.
- Apriori approaches
  - Generation of subgraph candidates is complicated and expensive.
  - Subgraph isomorphism is an NP-complete problem, so pruning is expensive.

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

FSG Algorithm

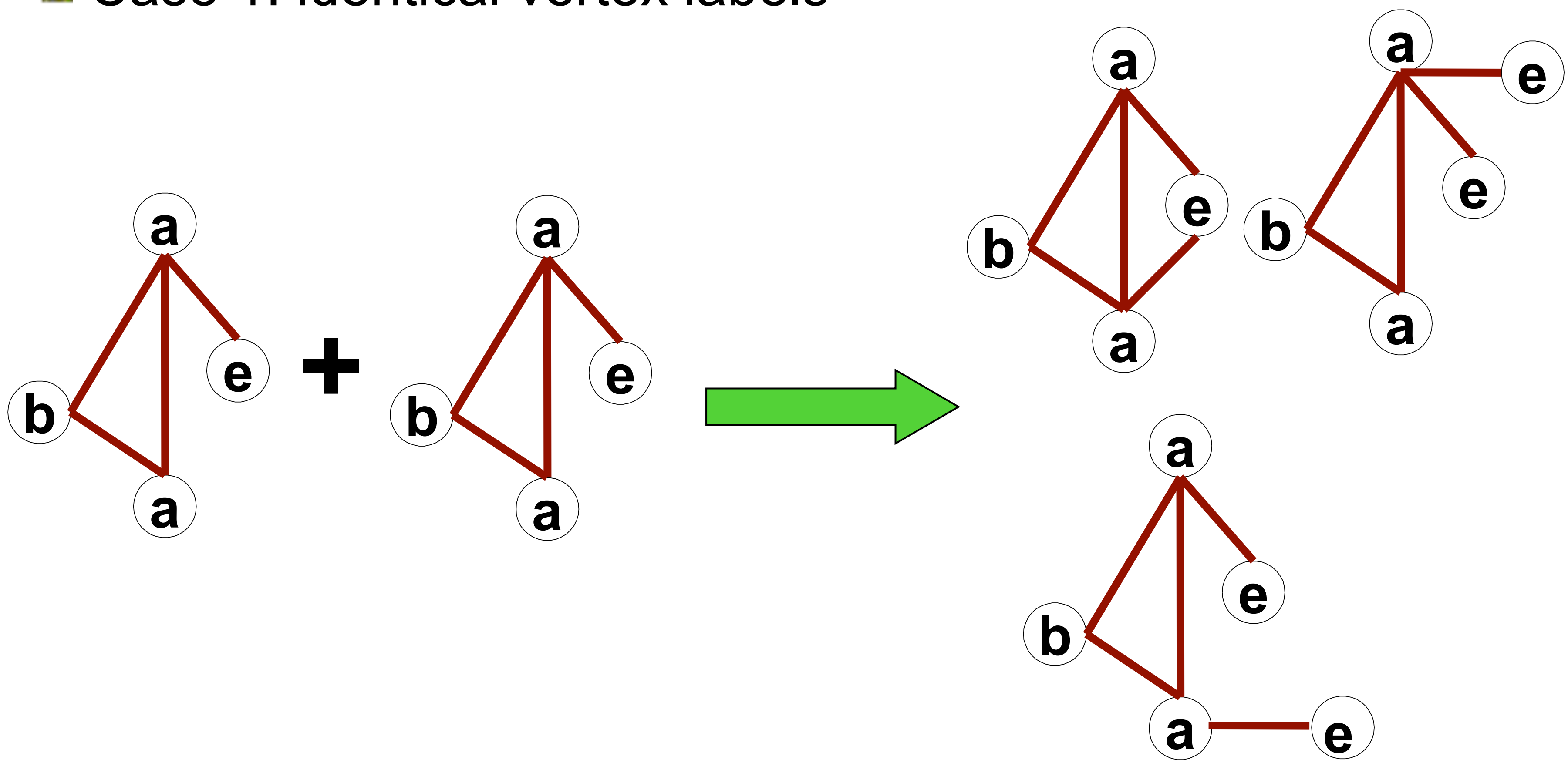
- $K = 1$
- F_1 = all frequent edges
- Repeat
  - $K = K + 1$;
  - $C_K = join(F_{K-1})$
  - F_K = frequent patterns in C_K
  - Until F_K is empty

Join: Key Operation

- Join(L) = $\cup$ join(P, Q) for all P, Q $\in$ L
- Join(P, Q) = {G | P, Q, $\subset$ G, |G| = |P| + 1, |P| = |Q|}
- Two graphs P and Q are ***joinable*** if the join of the two graphs produces an non-empty set
- Theorem: two graphs P and Q are joinable if P $\cap$ Q is a graph with size |P| -1 or share a common "core" with size P-1

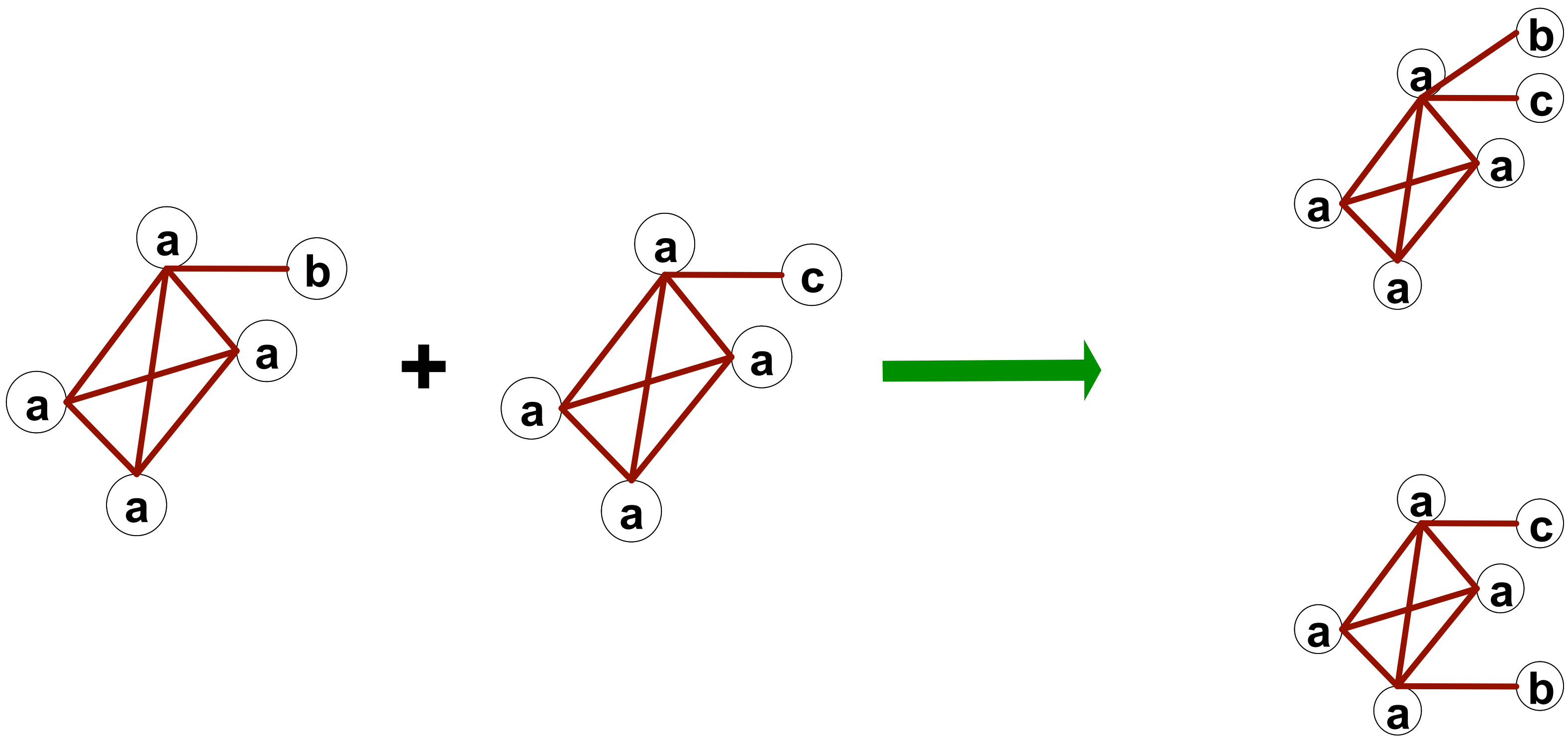
Multiplicity of Candidates

- Case 1: identical vertex labels



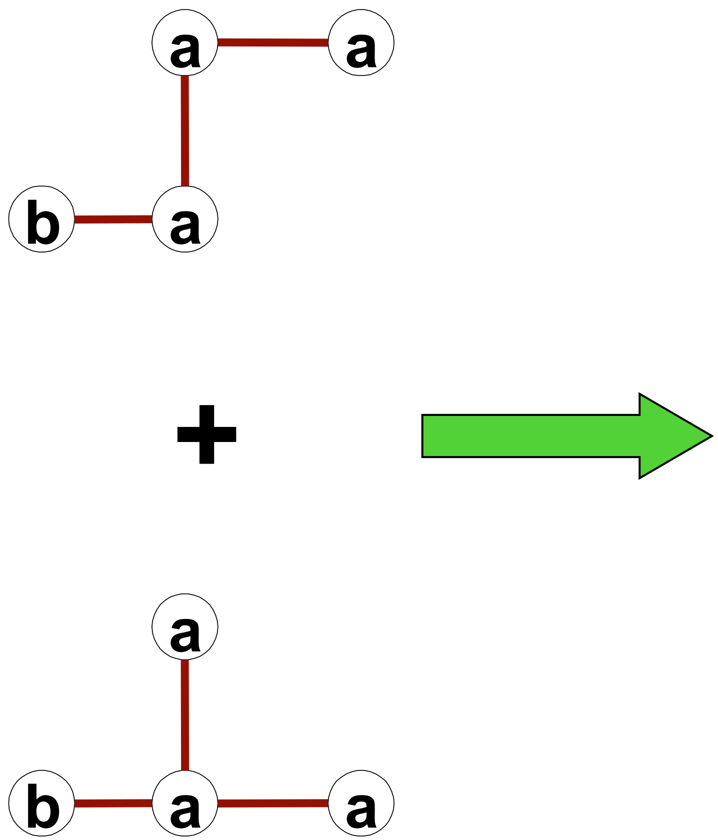
Multiplicity of Candidates

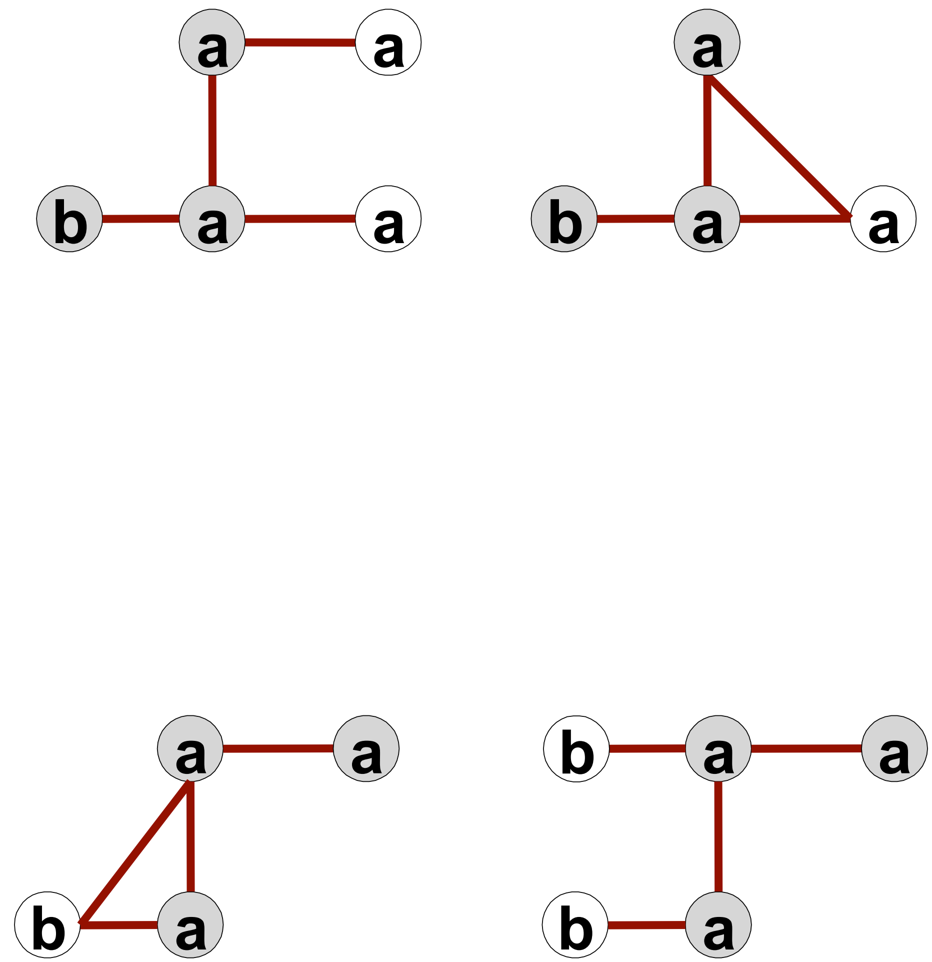
- Case 2: Core contains identical labels



Multiplicity of Candidates

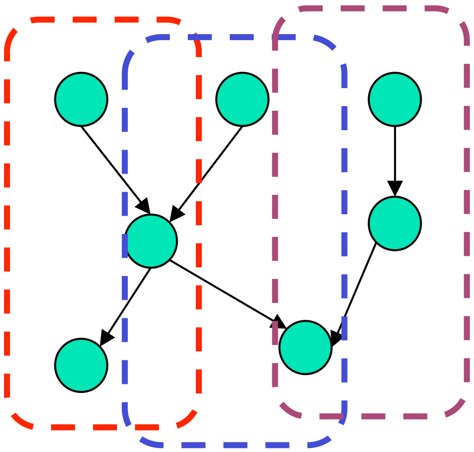
- Case 3: Core multiplicity





PATH

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



A graph with 3 edge-disjoint paths

PATH Algorithm

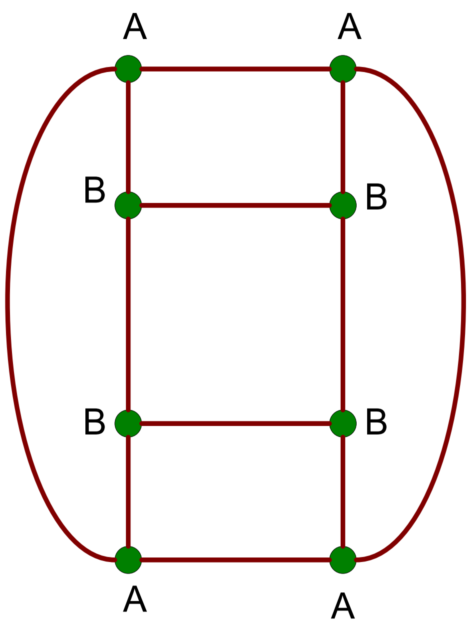
- $K = 1$
- F_1 = all frequent paths
- Repeat
  - $K = K + 1$;
  - $C_K = \text{join}(F_{K-1})$
  - F_K = frequent patterns in C_K
  - Until F_K is empty

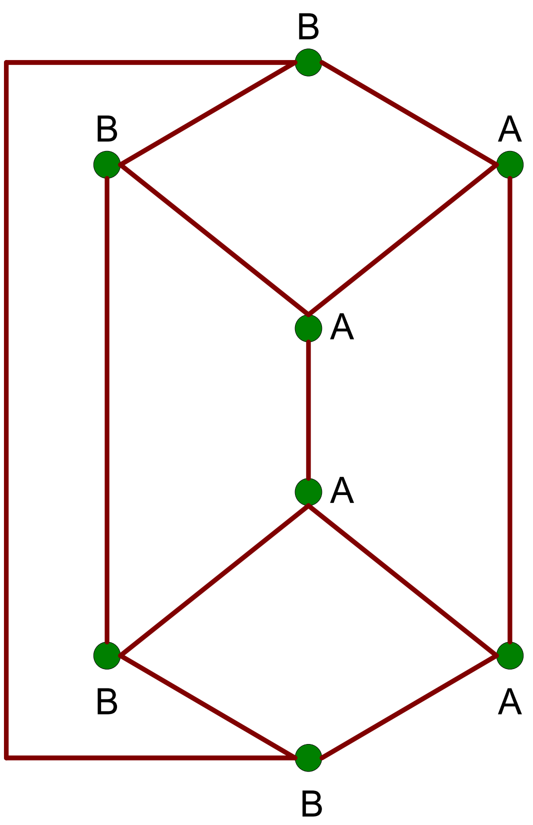
Challenges

- Graph isomorphism
  - Two graphs may have the same topology though their layouts are different
- Subgraph isomorphism
  - How to compute the support value of a pattern

Graph Isomorphism

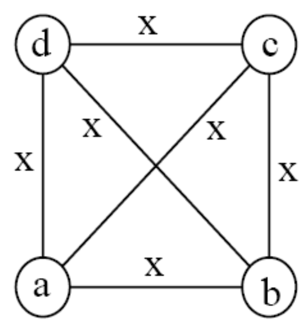
- A graph is isomorphic if it is topologically equivalent to another graph





Why Redundant Candidates?

- All the algorithms may propose the same candidate several times.

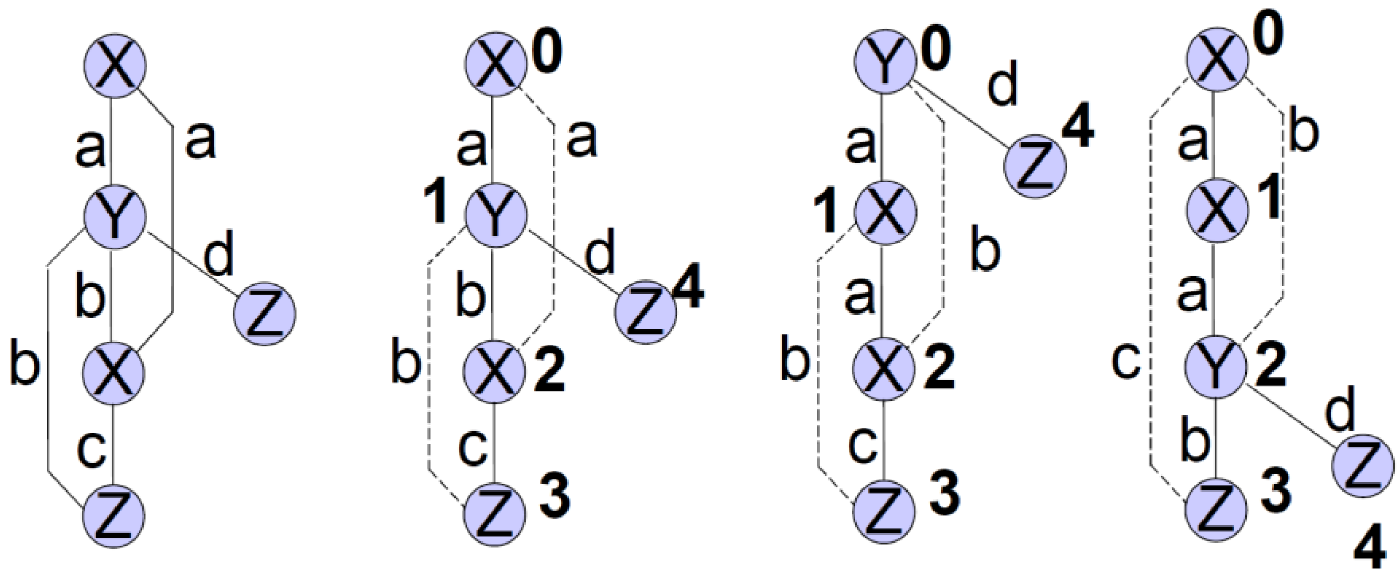


- We need to keep track of the identical candidates to
  - Avoid redundancy in results
  - Avoid redundant search

gSpan

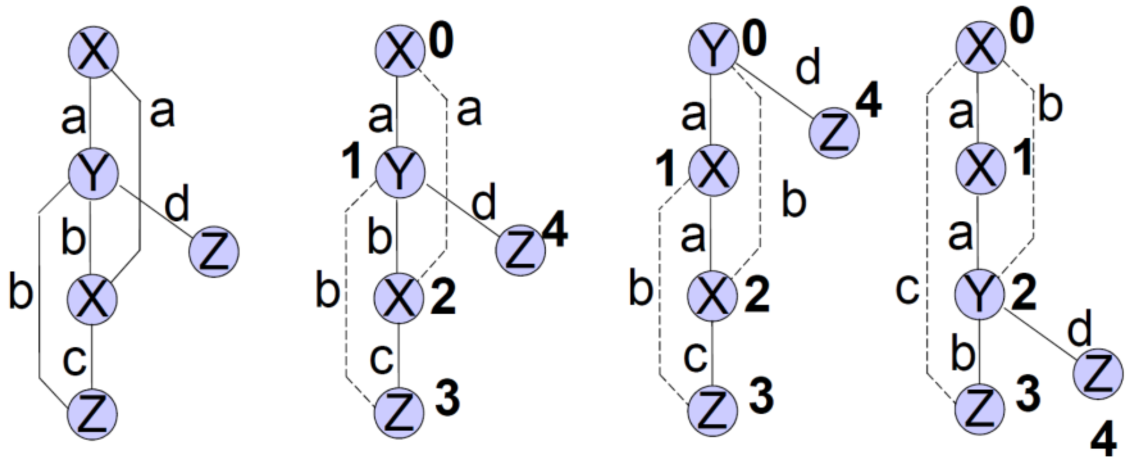
- DFS without candidate generation
  - Relabels graph representation to support DFS.
  - Discovers all frequent subgraphs without candidate generation or pruning.
- DFS Representation
  - Map each graph to a DFS code (sequence).
  - Lexicographically order the codes.
  - Construct a search tree based on the lexicographic order.

Depth-First Search Tree



(a) (b) (c) (d)

DFS Codes



(a) (b) (c) (d)

- Given $e_i = (i_1,j_1)$, $e_2 = (i_2,j_2)$: $e_1 < e_2$ if:

~$i_1 = i_2$ && $j_1 < j_2$

~$i_1 < j_1$ && $j_1 = i_2$

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

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)

DFS Lexicographic Order

- **Given lexicographic ordering of label set L, $\prec_L$**
- **Given graphs G_α, G_β (equivalent label sets).**
- **Given DFS codes**
  - $\alpha = \text{code}(G_\alpha, T_\alpha) = (a_0, a_1, \dots, a_m)$
  - $\beta = \text{code}(G_\beta, T_\beta) = (b_0, b_1, \dots, b_n)$
  - (assume $n \geq m$)
- $\boldsymbol{\alpha \leq \beta}$ **iff either of the following are true:**
  - $\exists t, 0 \leq t \leq \min(n, m)$ such that
    - $a_k = b_k$ for $k < t$ and
    - $a_t \prec_e b_t$
  - $a_k = b_k \; for \; 0 \leq k \leq m$

DFS Lexicographic Order

- **Given DFS codes**
  - $\alpha = \text{code}(G_\alpha, T_\alpha) = (a_0, a_1, \dots, a_m)$
  - $\beta = \text{code}(G_\beta, T_\beta) = (b_0, b_1, \dots, b_n)$
  - (assume $n \geq m$)
- **Given t such that** $a_k = b_k$ for $k < t$
- **Given** $\boldsymbol{a_t = (i_a, j_a, L_{i_a}, L_{i_a j_a}, L_{j_a})}$,
  $\boldsymbol{b_t = (i_b, j_b, L_{i_b}, L_{i_b j_b}, L_{j_b})}$,
- $\boldsymbol{a_t \prec_e b_t}$ **if one of the following cases**

Case 1:
Both forward edges, AND…

Case 2:
Both back edges, AND…

Case 3: a_t back, b_t forward → $a_t \prec_e b_t$

DFS Lexicographic Order (Case 1)

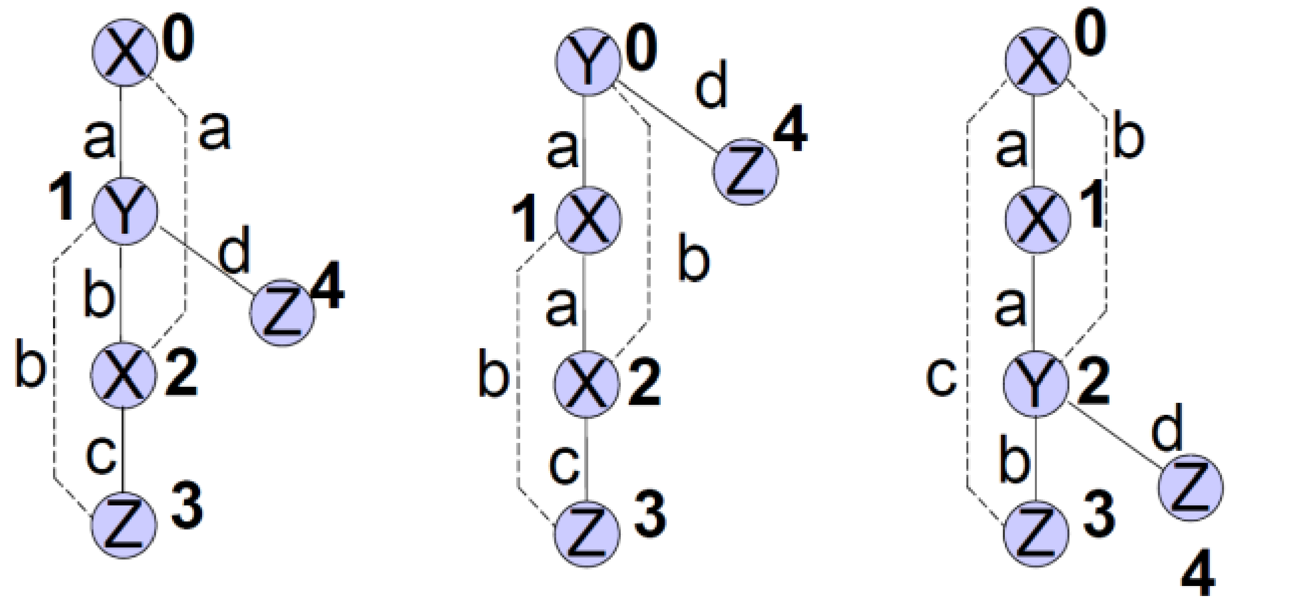
- **Both forward edges, AND one of the following:**
  - $i_b < i_a$ (edge starts from a ***later*** visited vertex)
    - Why is this (think about DFS process)?
  - $i_a = i_b$ AND labels of a lexicographically less than labels of b, in order of tuple.
    - Ex: Labels are strings, $a_t = (\_, \_, \mathrm{m}, \mathrm{e}, \mathrm{x})$, $b_t = (\_, \_, \mathrm{m}, \mathrm{u}, \mathrm{x})$
      - m = m, e < u → $a_t \prec_e b_t$
- **Note: if both forward edges, then $j_a = j_b$**
  - Reasoning: all previous edges equal, target vertex discovery times are the same

DFS Lexicographic Order (Case 2)

- **Both back edges, AND one of the following:**
  - $j_a < j_b$ (edge refers to earlier vertex)
  - $j_a = j_b$ AND edge label of a lexicographically less than b
    - Note: given that all previous edges equal, vertex labels must also be equal
- **Note: if both back edges, then $i_a = i_b$**
  - Reasoning: all previous edges equal, source vertex discovery times are the same.

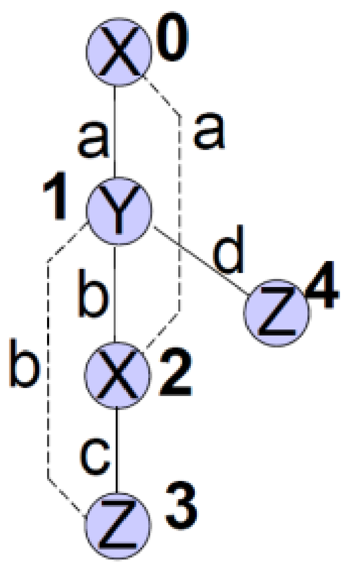
If X<Y<Z and a<b<c, then code C<A<B

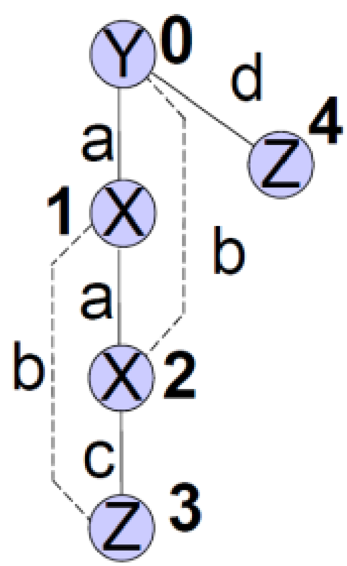
	A	B	C
0	(0,1,X,a,Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	(1,2,X,a,X)	(1,2,X,a,Y)
2	(2,0,X,a,X)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	(3,1,Z,b,X)	(3,0,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)

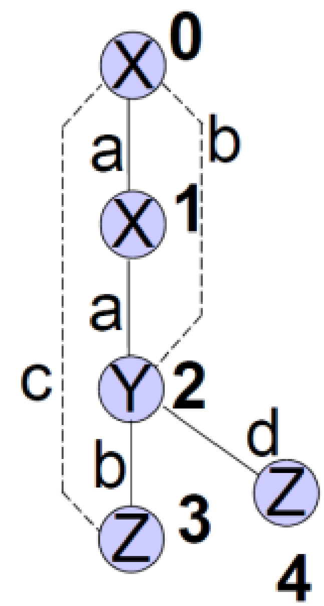


If X=Y=Z and a=b=c, then code C<A<B

	A	B	C
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)	(3,1,Z,b,X)	(3,0,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)





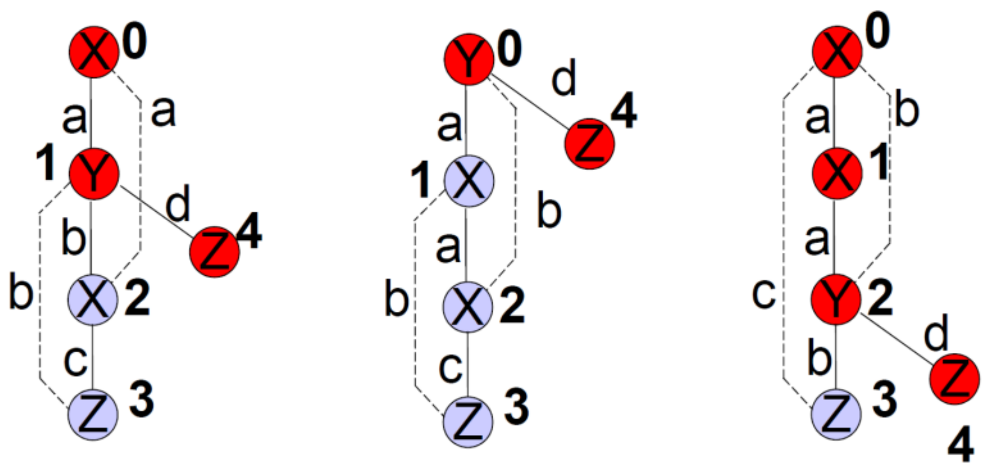


DFS Lexicographic Order

- 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)*.

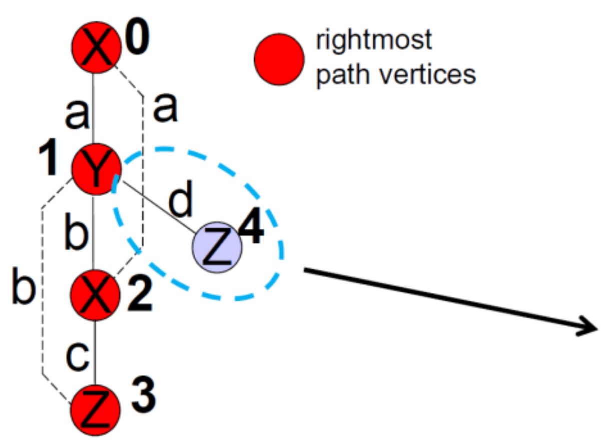
DFS Codes: Parents and Children

- If ∂ = $(a_0, a_1, \ldots, a_m)$ and ß = $(a_0, a_1, \ldots, a_m, b)$:
  - ß is the child of ∂.
  - ∂ is the parent of ß.
- A valid DFS code requires that b grows from a vertex on the rightmost path.
  - Rightmost vertex: v_n
  - Rightmost path: shortest path from v_0 to v_n using only forward edges



DFS code extension

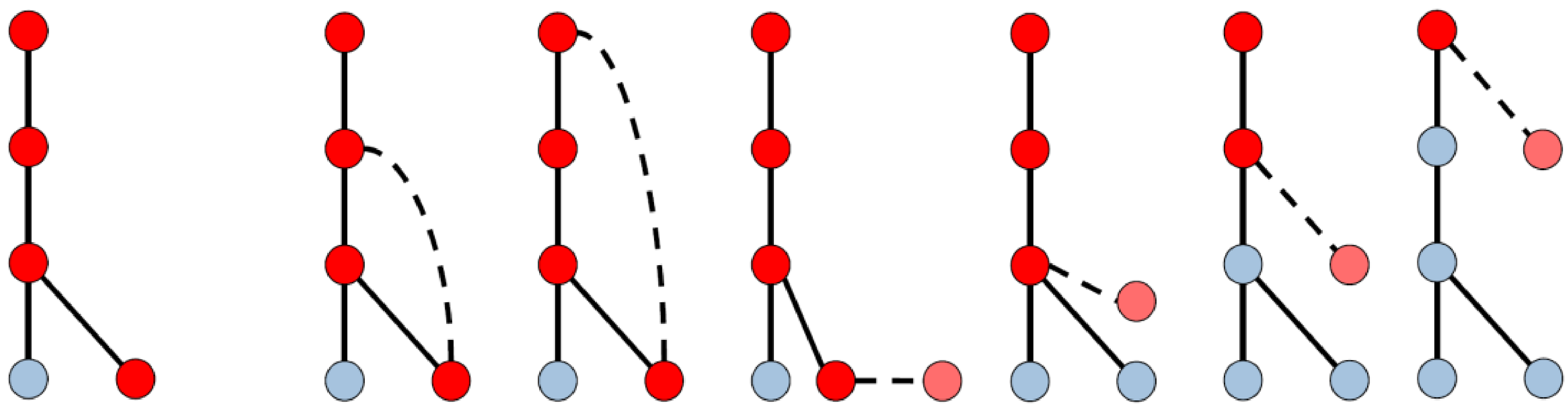
- Forward edge extensions to a DFS code must occur from a vertex on the rightmost path!



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

DFS code extension

- Back edge extensions must occur from the rightmost vertex!



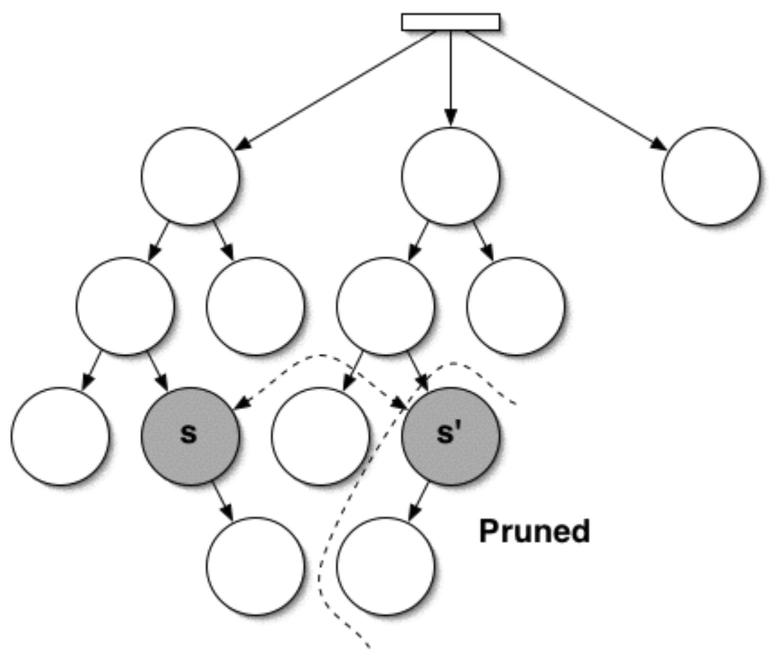
DFS code extension

- if a vertex not on rightmost path, then it has been fully processed by DFS.

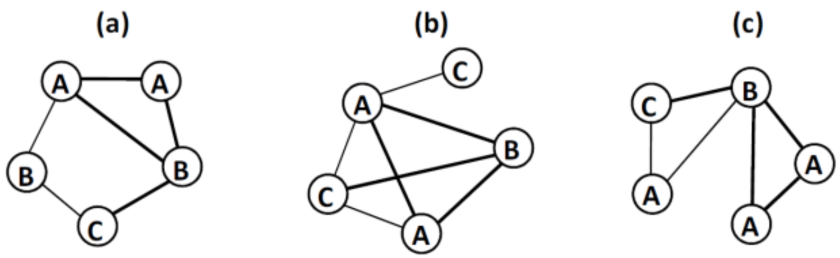
- previous last DFS edge tuple $<$ new tuple, if
  - new edge is forward, extended from a vertex on rightmost path, OR
  - new edge is backward, extended from rightmost vertex

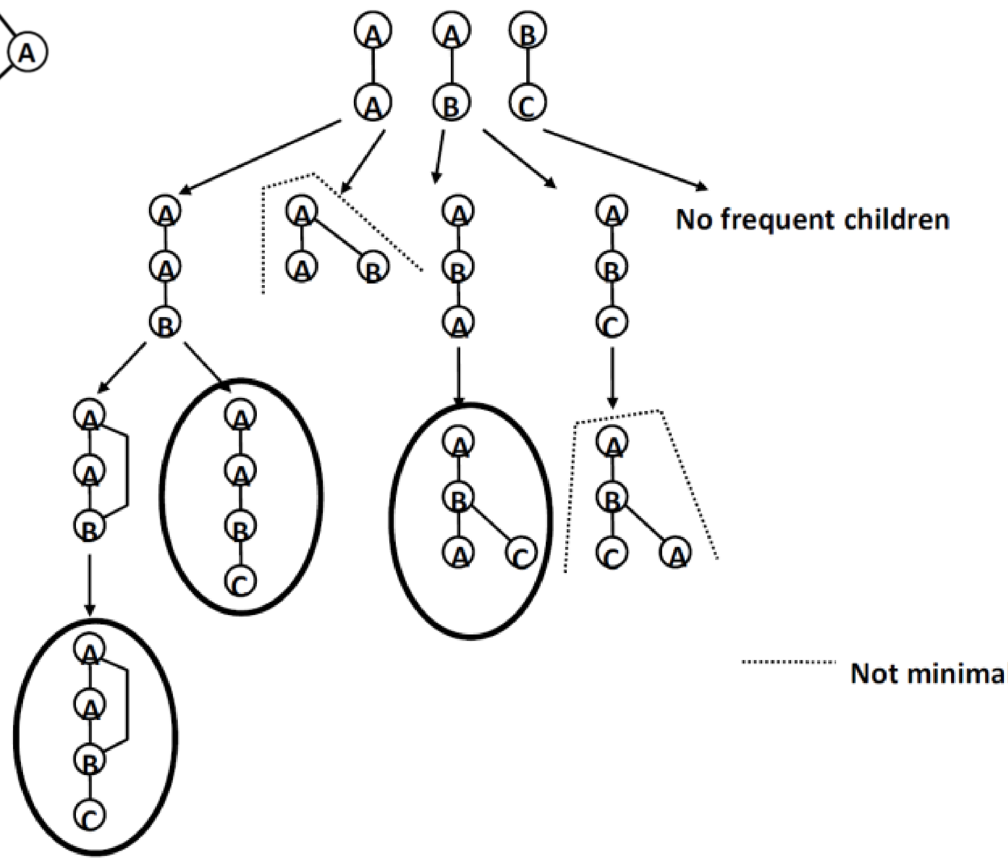
DFS Code Trees

- Organize DFS code nodes as parent-child.
- Pre-order traversal follows DFS lexicographic order.
- If s and s’ are the same graph with different DFS codes, s’ is not the minimum and can be pruned.



Example





Min_support = 3

gSpan

- D is the set of all graphs.
- S is the result set.

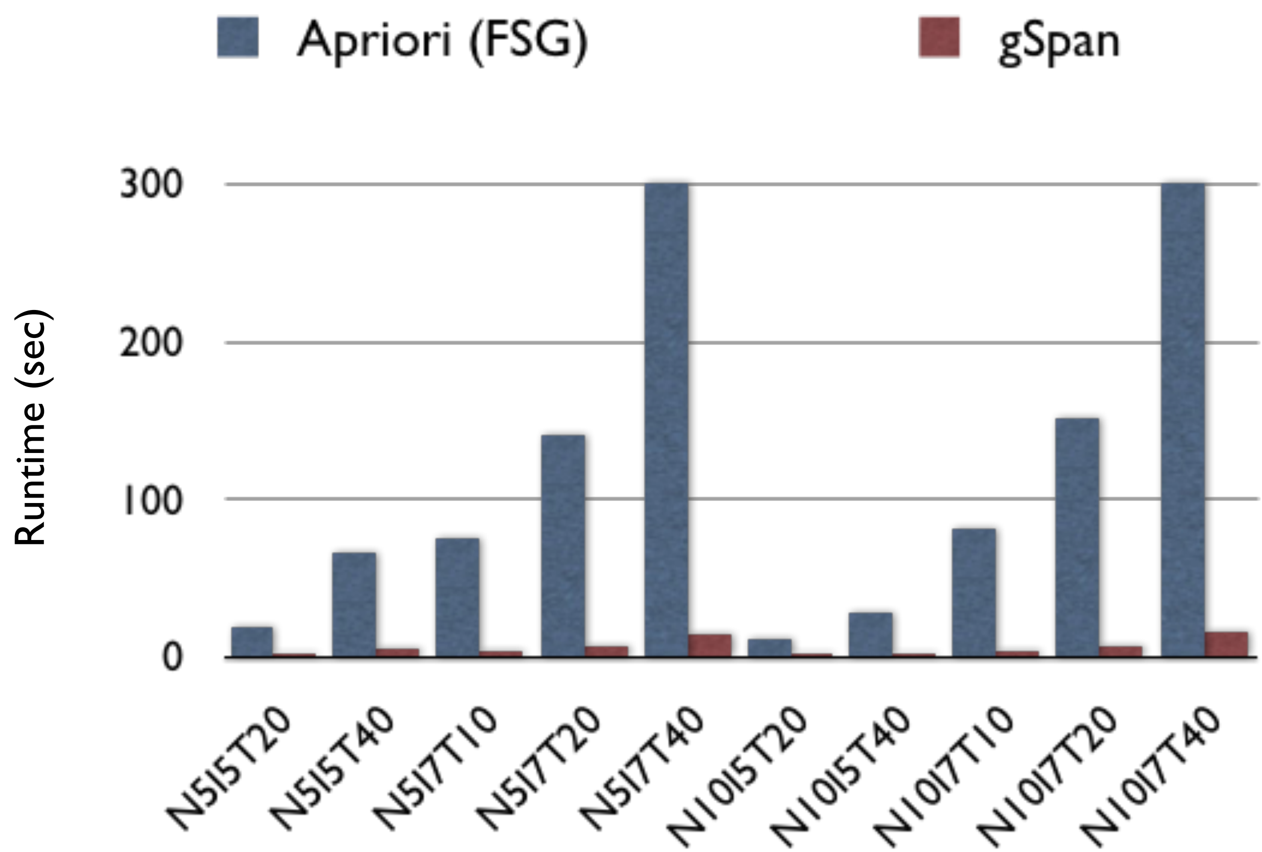
Algorithm 1: GraphSet_Projection(*D*,*S*)

```
sort labels in D by frequency
remove infrequent vertices and edges
relabel remaining vertices and edges
S' = all frequent 1-edge graphs in D
sort S' in DFS lexicographic order
S = S'
foreach edge e in S' do
  s = graph defined by e
  s.D = subgraphs in D containing e
  Subgraph_Mining(D,S,s)
  D = D - e
  if |D| < minSup
        break
```

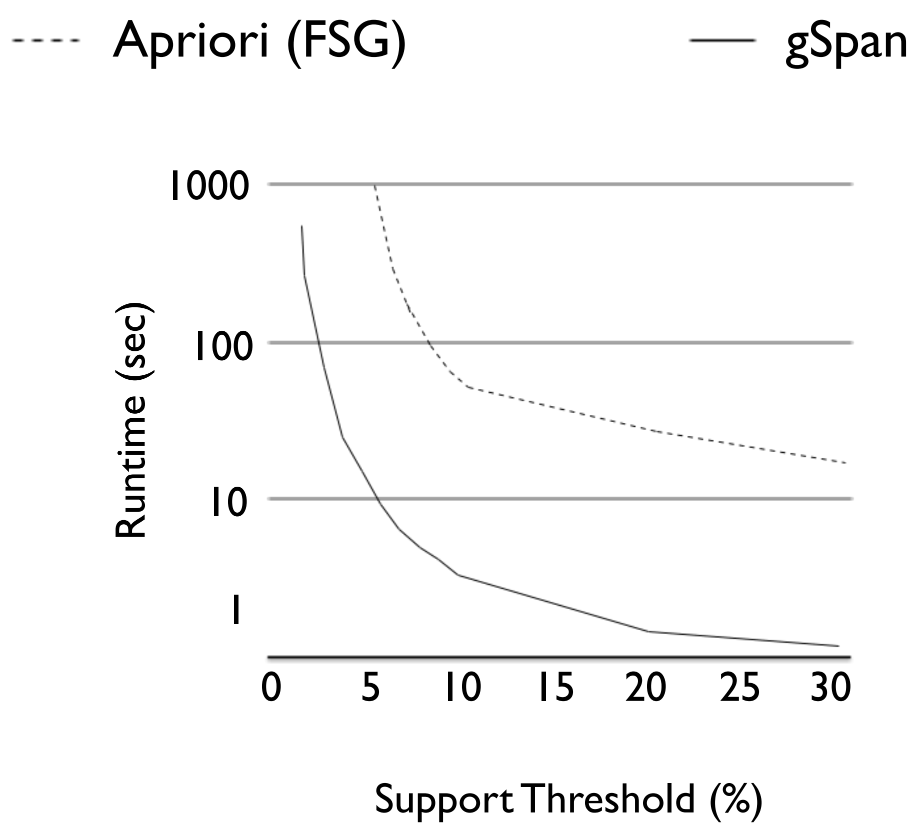
Subprocedure 1: Subgraph_Mining(*D*,*S*,*s*)

```
if s != min(s)
  return
S = S U {s}
s' = +1-edge children of s in s.D
foreach child c of s' do
  if support(c) ≥ minSup
        Subgraph_Mining(D_s,S,c)
```

Runtime: Synthetic



Runtime: Chemical



gSpan Advantages

- Lower memory requirements.
- Faster than naïve FSG by an order of magnitude.
- No candidate generation.
- Lexicographic ordering minimizes search tree.
- False positives pruning.

- Any disadvantage?