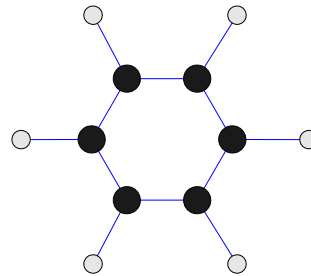
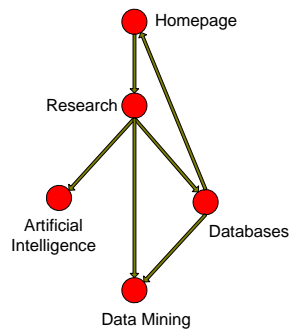
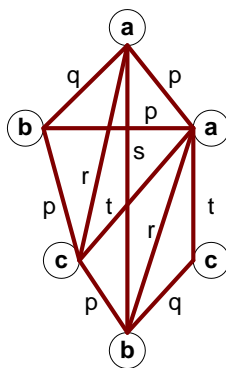


Frequent Subgraph Mining

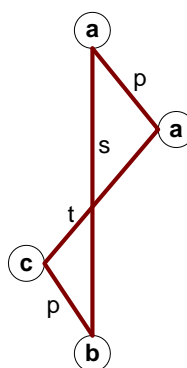
- Extend association rule mining to finding frequent subgraphs
- Useful for Web Mining, computational chemistry, bioinformatics, spatial data sets, etc



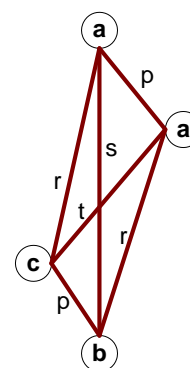
Graph Definitions



(a) Labeled Graph



(b) Subgraph

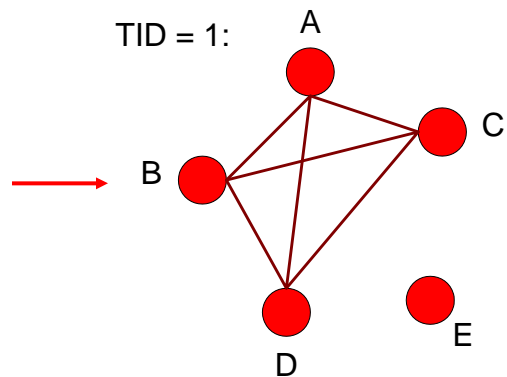


(c) Induced Subgraph

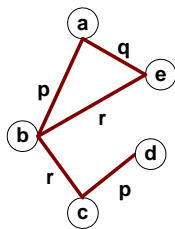
Representing Transactions as Graphs

- Each transaction is a clique of items

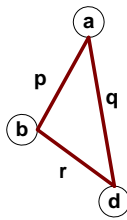
Transaction Id	Items
1	{A,B,C,D}
2	{A,B,E}
3	{B,C}
4	{A,B,D,E}
5	{B,C,D}



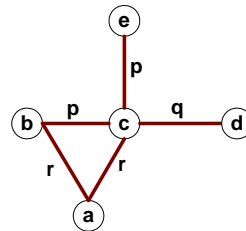
Representing Graphs as Transactions



G1



G2



G3

	(a,b,p)	(a,b,q)	(a,b,r)	(b,c,p)	(b,c,q)	(b,c,r)	...	(d,e,r)
G1	1	0	0	0	0	1	...	0
G2	1	0	0	0	0	0	...	0
G3	0	0	1	1	0	0	...	0
G3

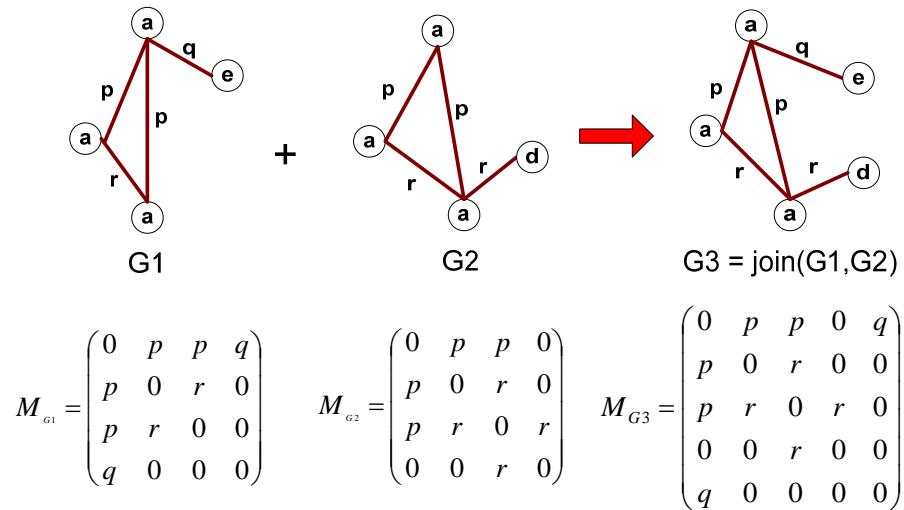
Challenges

- Node may contain duplicate labels
- Support and confidence
 - How to define them?
- Additional constraints imposed by pattern structure
 - Support and confidence are not the only constraints
 - Assumption: frequent subgraphs must be connected
- Apriori-like approach:
 - Use frequent k -subgraphs to generate frequent $(k+1)$ subgraphs
 - ◆ What is k ?

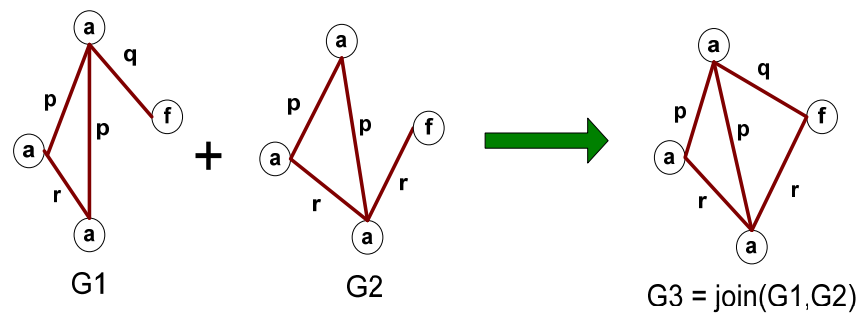
Challenges...

- Support:
 - number of graphs that contain a particular subgraph
- Apriori principle still holds
- Level-wise (Apriori-like) approach:
 - Vertex growing:
 - ◆ k is the number of vertices
 - Edge growing:
 - ◆ k is the number of edges

Vertex Growing



Edge Growing

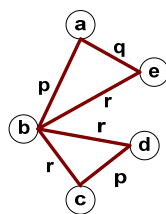


Apriori-like Algorithm

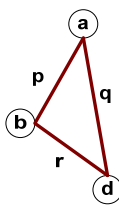
- Find frequent 1-subgraphs
- Repeat
 - Candidate generation
 - ◆ Use frequent $(k-1)$ -subgraphs to generate candidate k -subgraph
 - Candidate pruning
 - ◆ Prune candidate subgraphs that contain infrequent $(k-1)$ -subgraphs
 - Support counting
 - ◆ Count the support of each remaining candidate
 - Eliminate candidate k -subgraphs that are infrequent

In practice, it is not as easy. There are many other issues

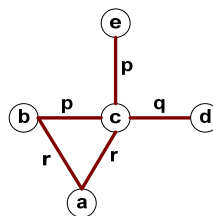
Example: Dataset



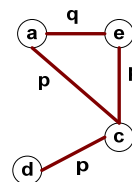
G1



G2



G3



G4

	(a,b,p)	(a,b,q)	(a,b,r)	(b,c,p)	(b,c,q)	(b,c,r)	...	(d,e,r)
G1	1	0	0	0	0	1	...	0
G2	1	0	0	0	0	0	...	0
G3	0	0	1	1	0	0	...	0
G4	0	0	0	0	0	0	...	0

Example

Minimum support count = 2

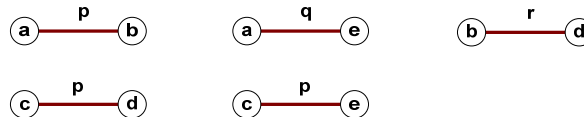
k=1

Frequent
Subgraphs



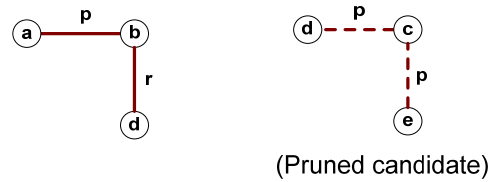
k=2

Frequent
Subgraphs



k=3

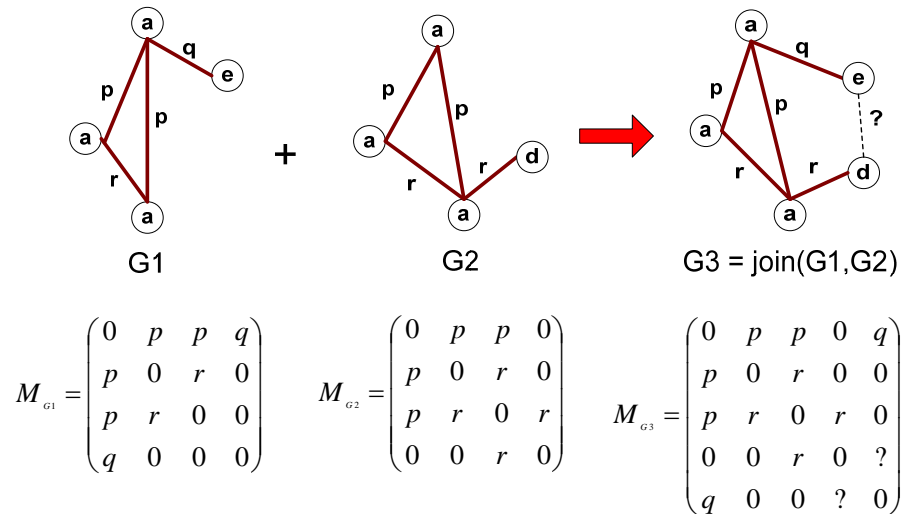
Candidate
Subgraphs



Candidate Generation

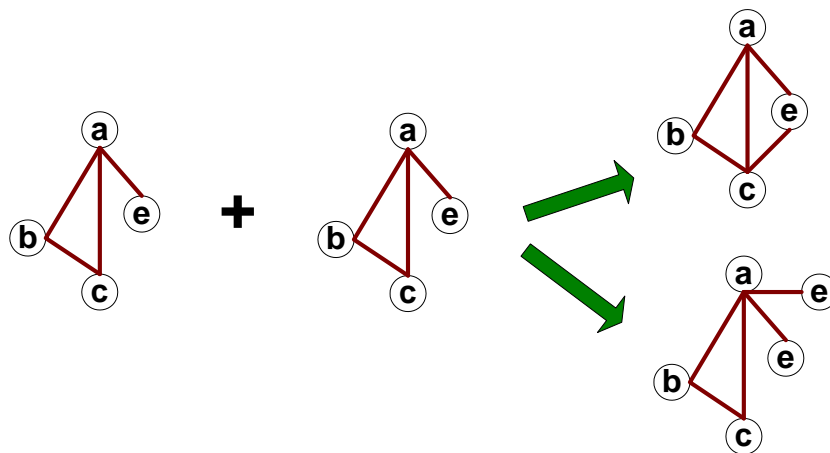
- In Apriori:
 - Merging two frequent k -itemsets will produce a candidate $(k+1)$ -itemset
- In frequent subgraph mining (vertex/edge growing)
 - Merging two frequent k -subgraphs may produce more than one candidate $(k+1)$ -subgraph

Multiplicity of Candidates (Vertex Growing)



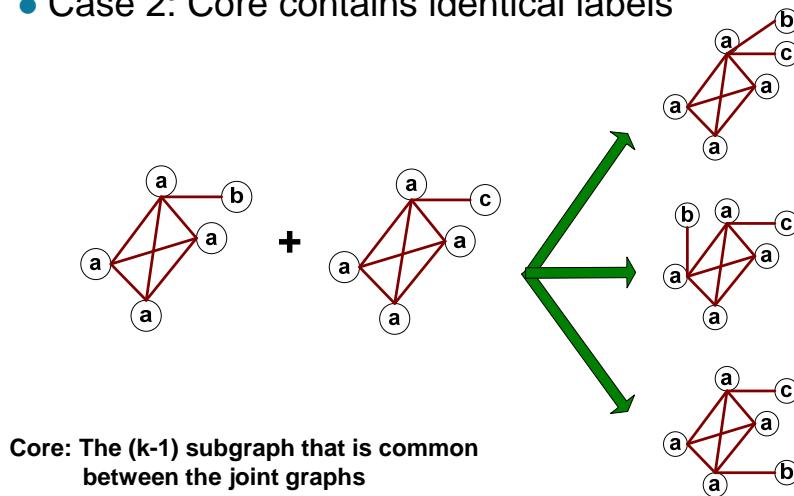
Multiplicity of Candidates (Edge growing)

- Case 1: identical vertex labels



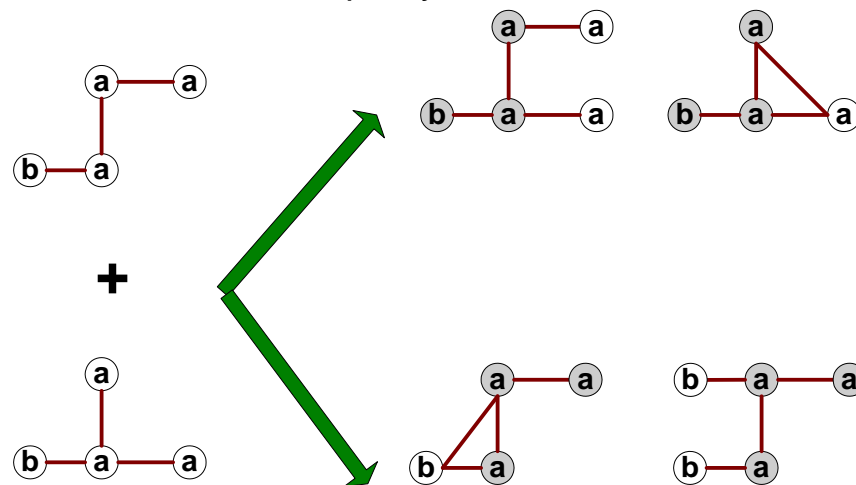
Multiplicity of Candidates (Edge growing)

- Case 2: Core contains identical labels

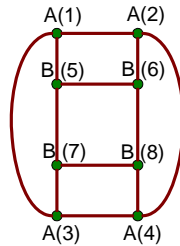


Multiplicity of Candidates (Edge growing)

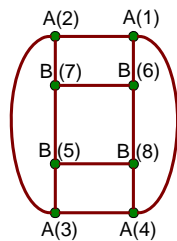
- Case 3: Core multiplicity



Adjacency Matrix Representation



	A(1)	A(2)	A(3)	A(4)	B(5)	B(6)	B(7)	B(8)
A(1)	1	1	1	0	1	0	0	0
A(2)	1	1	0	1	0	1	0	0
A(3)	1	0	1	1	0	0	1	0
A(4)	0	1	1	1	0	0	0	1
B(5)	1	0	0	0	1	1	1	0
B(6)	0	1	0	0	1	1	0	1
B(7)	0	0	1	0	1	0	1	1
B(8)	0	0	0	1	0	1	1	1

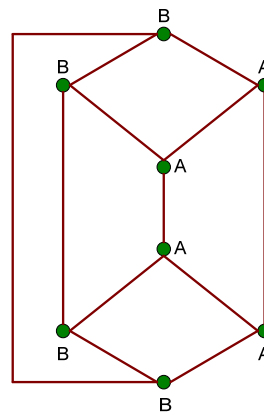
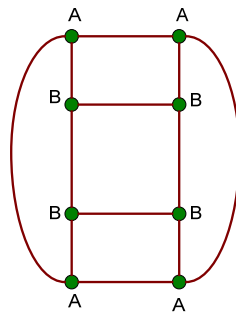


	A(1)	A(2)	A(3)	A(4)	B(5)	B(6)	B(7)	B(8)
A(1)	1	1	0	1	0	1	0	0
A(2)	1	1	1	0	0	0	1	0
A(3)	0	1	1	1	1	0	0	0
A(4)	1	0	1	1	0	0	0	1
B(5)	0	0	1	0	1	0	1	1
B(6)	1	0	0	0	0	1	1	1
B(7)	0	1	0	0	1	1	1	0
B(8)	0	0	0	1	1	1	0	1

- The same graph can be represented in many ways

Graph Isomorphism

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



Graph Isomorphism

- Test for graph isomorphism is needed:
 - During candidate generation step, to determine whether a candidate has been generated
 - During candidate pruning step, to check whether its $(k-1)$ -subgraphs are frequent
 - During candidate counting, to check whether a candidate is contained within another graph

Graph Isomorphism

- Use canonical labeling to handle isomorphism
 - Map each graph into an ordered string representation (known as its code) such that two isomorphic graphs will be mapped to the same canonical encoding
 - Example:
 - ◆ Lexicographically largest adjacency matrix

