HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

582364 Data mining, 4 cu
Lecture 8:
Graph mining

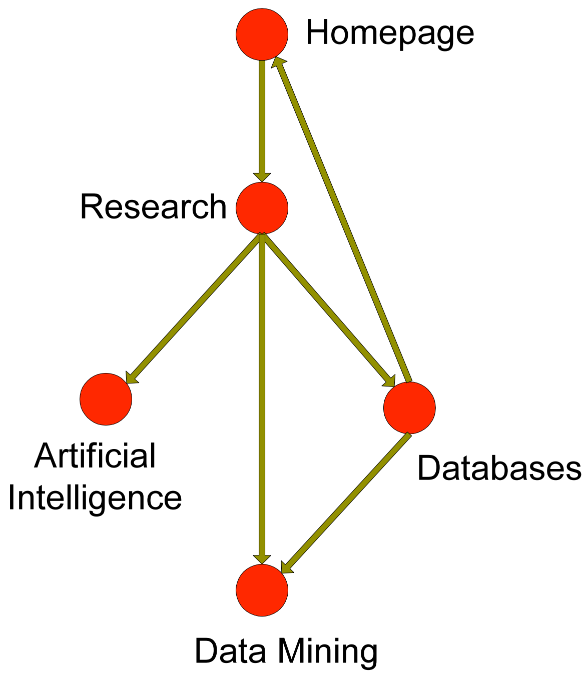
Spring 2010

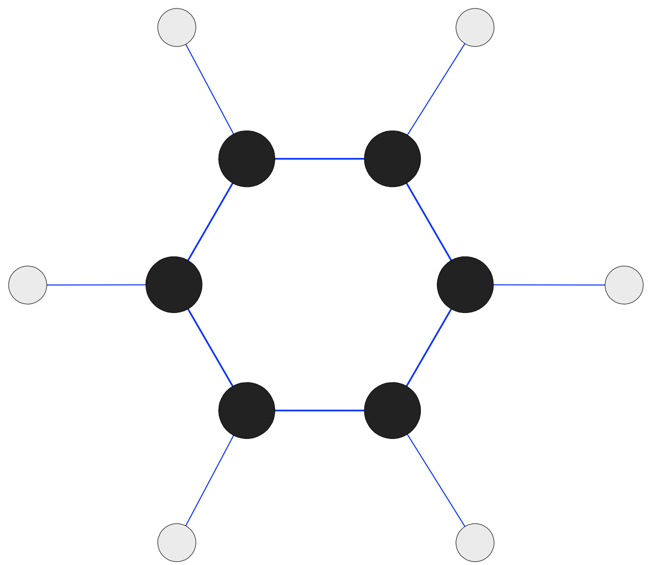
Lecturer: Juho Rousu

Teaching assistant: Taru Itäpelto

Frequent Subgraph Mining

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



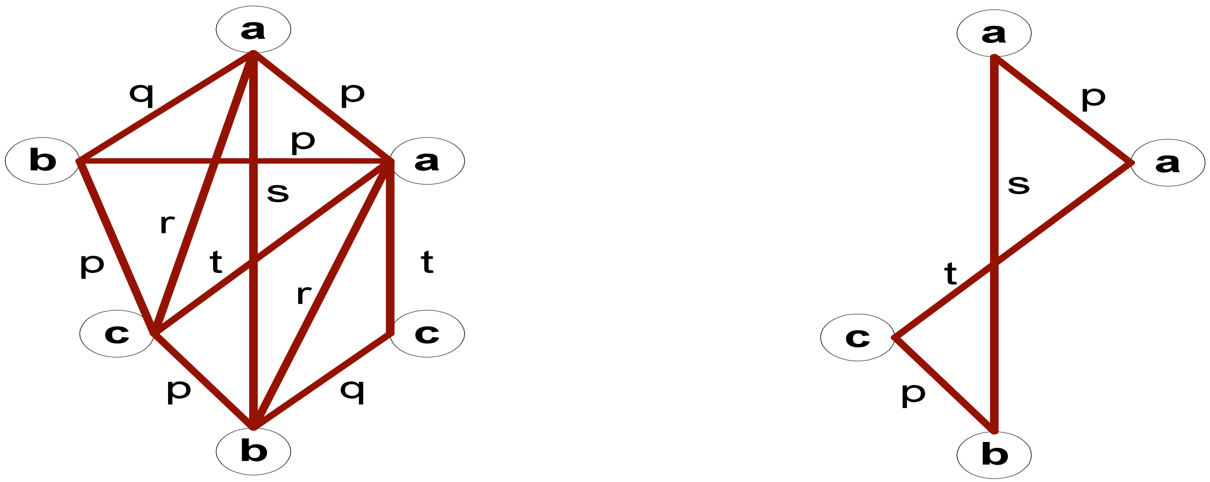


Graphs in Applications

Application	Graphs used to analyze	Vertices	Edges
Web mining	Web browsing patterns	Web pages	Hyperlinks between pages
Computational chemistry	Structure of chemical compounds	Atoms	Bonds
Networking	Internet routing	Server computers	Interconnection between servers
Bioinformatics	Gene/protein interaction	Genes/proteins	Regulatory relations, physical binding

Graphs and subgraphs

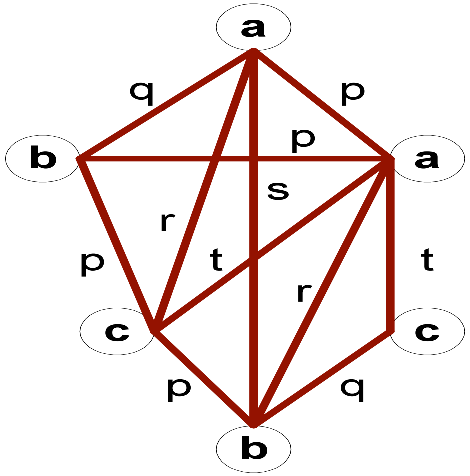
- A graph G = (V,E) is composed of vertices (nodes) V and a set of edges E.
- In labeled graphs, vertices, edges or both can have labels describing them and differentiating them from each other
- Graph G' =(V' ,E') is a subgraph of G = (V,E) if V' is a subset of V and E' is a subset of E



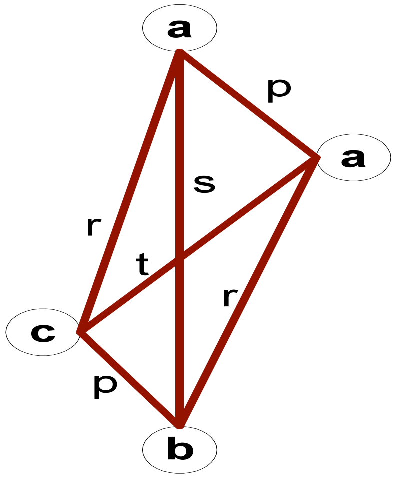
(a) Labeled Graph (b) Subgraph

Induced subgraph

- Graph G' =(V' ,E') is an induced subgraph of G = (V,E) if
  - V' is a subset of V,
  - E' contains all edges in E that have both ends in the set V'
- The number of induced subgraphs is typically significantly less than the number of general subgraphs
  - in induced subgraph E' is determined by G and V'



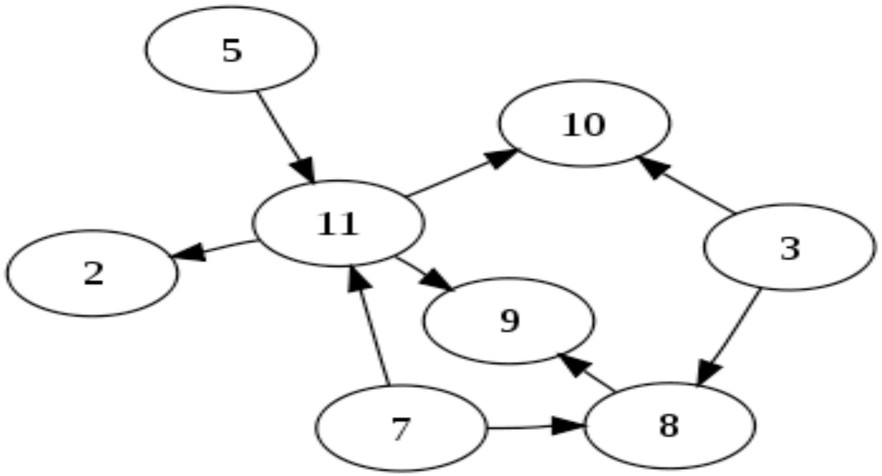
(a) Labeled Graph

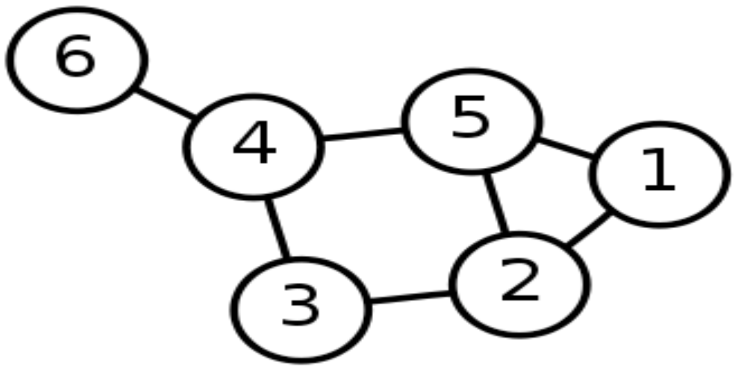


(c) Induced Subgraph

Directed and undirected graphs

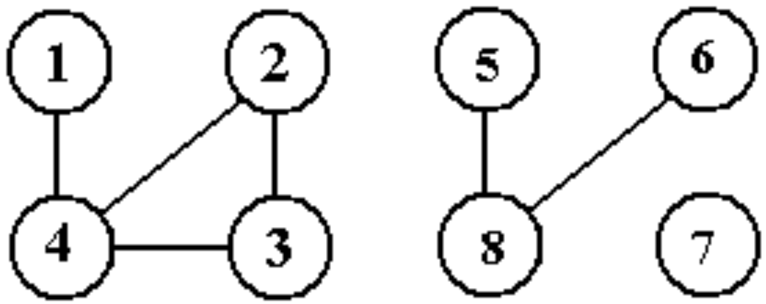
- Graph is *directed* if the edges are *oriented* (denoted by arrowhead),
  - i.e. edge (u,v) is different object from edge (v,u)
- Graph is undirected if edges have no orientation
  - (u,v) and (v,u) denote the same object
- We concentrate in undirected graphs





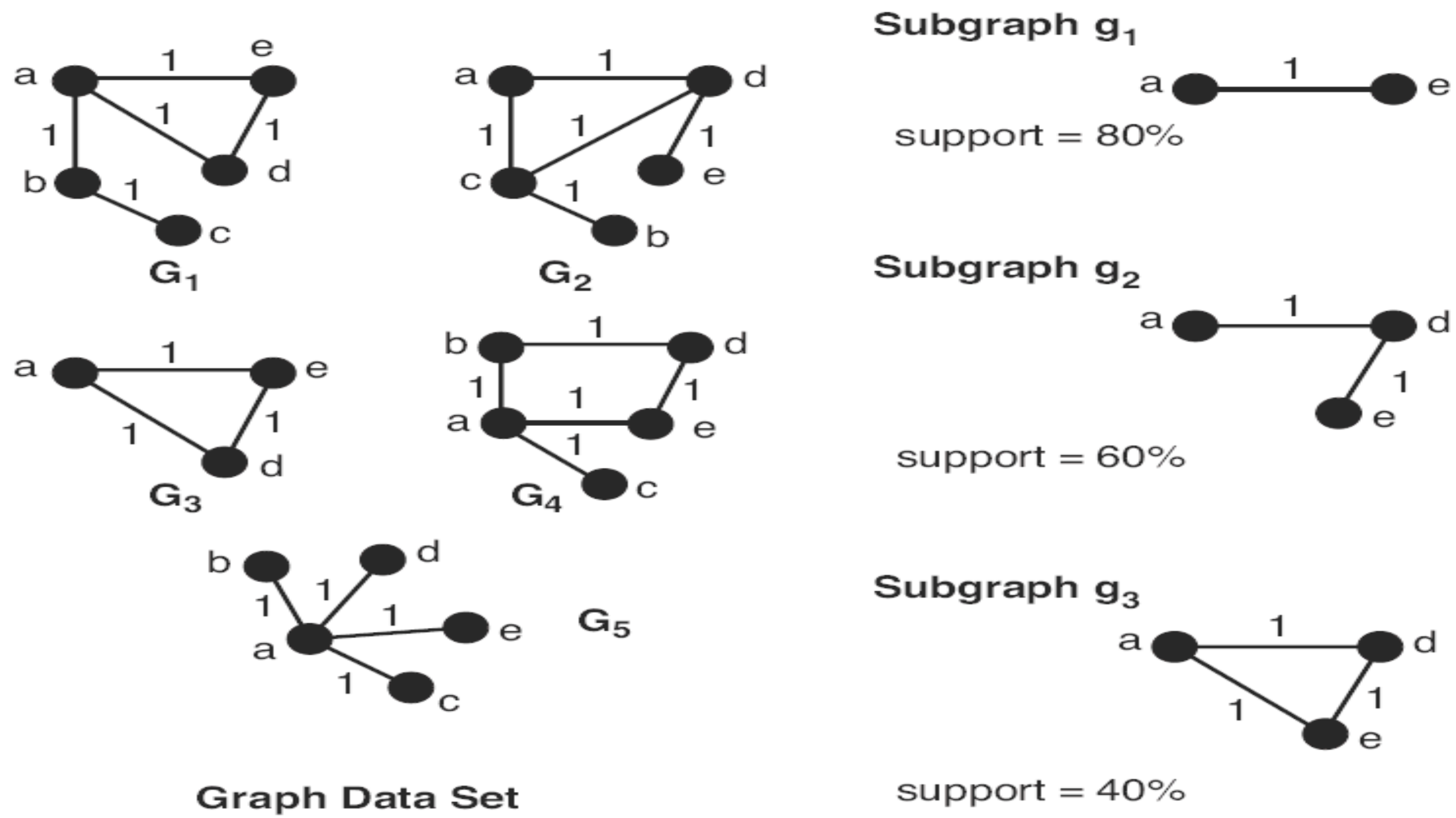
Connected and disconnected graphs

- Graph G = (V,E) is connected, if there is a path between any two nodes in V
- Otherwise the graph is disconnected
- A connected component is a maximal connected subgraph of a graph
  - below, {1,2,3,4}, {5,8,6} and {7} with the connecting edges
- We concentrate in connected graphs



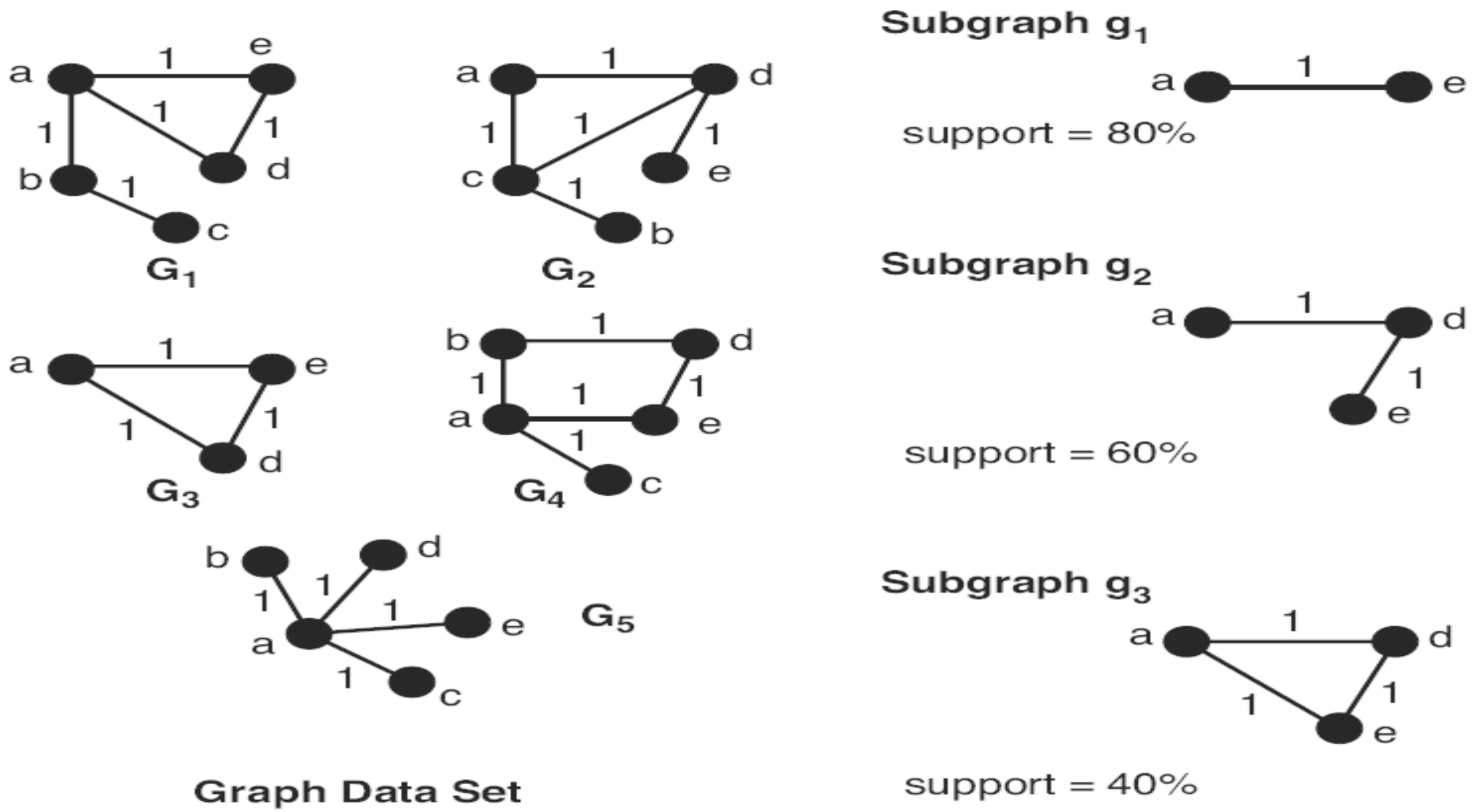
Support of a subgraph

- Given a collection of graphs **G**, the support of subgraph g is defined as the fraction of all graphs that contain g as its subgraph



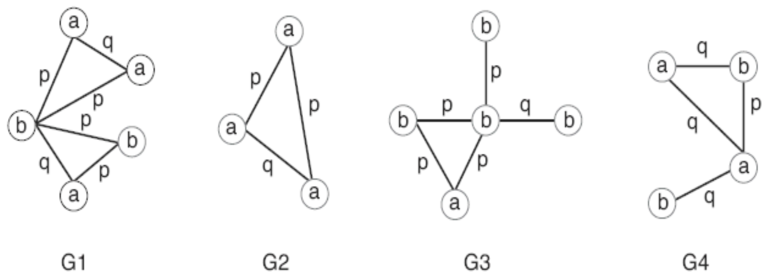
Frequent subgraph mining: Definition

- Given a set of graphs **G** and a support threshold *minsup*, the goal is to find all subgraphs g with support s(g) at least *minsup*

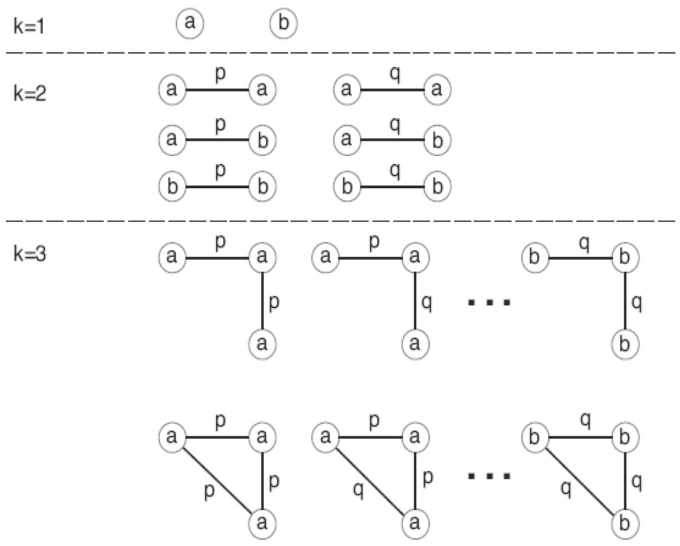


Brute-force method?

- Generate all connected subgraphs, count the supports, and prune
- Problem: exponential number of subgraphs
- Considerably higher number of subgraphs than itemsets, given the same items (=node labels)
  - An item can appear only once in an itemset but many times in a subgraph
  - Given a fixed set of nodes, edges can be organized and labeled in many ways to create a set of different subgraphs
  - Also more subgraphs than subsequences



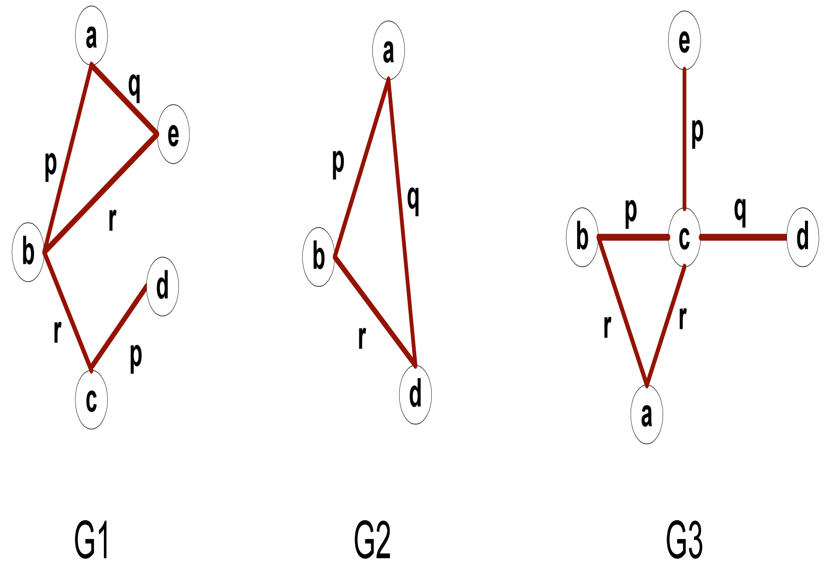
(a) Example of a graph data set.



(b) List of connected subgraphs.

Using itemset Apriori for subgraph mining

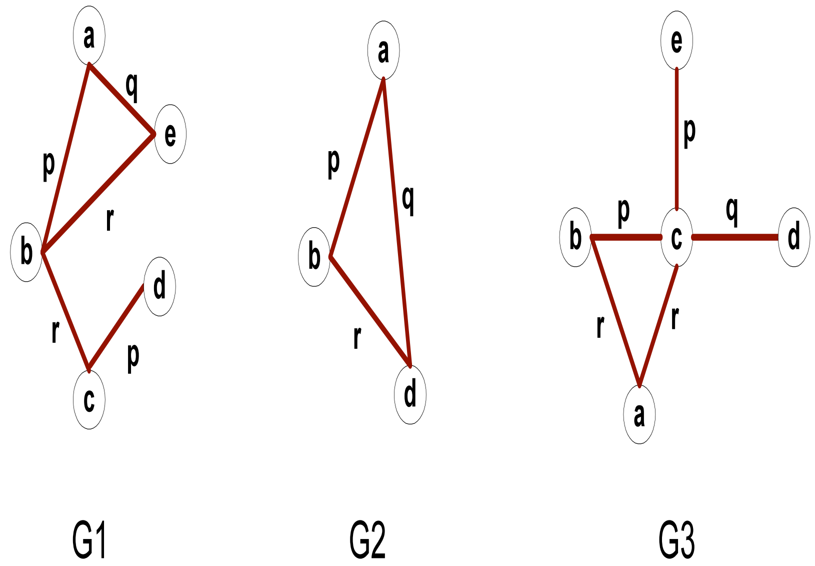
- One approach to mine subgraphs efficiently is to transform the graph dataset into a transaction database
  - Each combination of vertex label – edge label – vertex label is defined as an item
  - The width of the transaction is the number of edges in the graph



	(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	...	...	...	...	...	...	...	...

Using itemset Apriori for subgraph mining

- Problem: Multiple edges will be mapped into one item if they have the same label combination
  - loss of information
- How to convert a frequent itemset into a frequent subgraph?
  - Which of the edges in the original graph to choose
  - Subgraph structure will be ambiquous

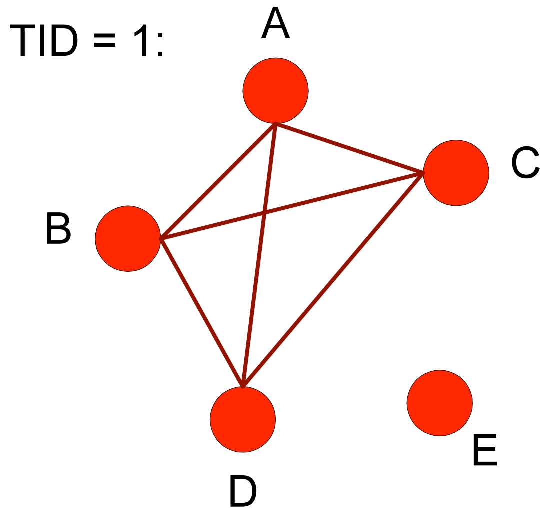


	(a,b,p)	(a,b,q)	(a,b,r)	(b,c,p)	(b,c,q)	(b,c,r)	...	(d,e,r)
G1	1	0	0	0	0	1	...	0
G2	1	0	0	0	0	0	...	0
G3	0	0	1	1	0	0	...	0
G3	...	...	...	...	...	...	...	...

Note: Representing Transactions as Graphs

- The other direction, mapping *transactions* to *graphs* does not lose information
- Each transaction is a clique (fully connected subgraph) of items, an itemset is a subset of the clique
- So frequent subgraph mining can solve frequent itemset mining (in principle), but not vice versa

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



Apriori-like approach for frequent subgraph mining

- Try to follow the usual Apriori scheme:

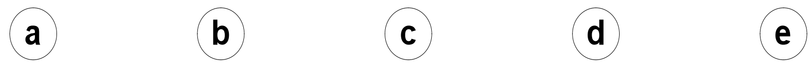
1. Find frequent 1-subgraphs
2. Repeat until no more frequent subgraphs are found;
   1. Candidate generation
      - Use frequent (*k-1*)-subgraphs to generate candidate *k*-subgraph
   2. Candidate pruning
      - Prune candidate subgraphs that contain infrequent (*k-1*)-subgraphs
   3. Support counting
      - Count the support of each remaining candidate
   4. Eliminate candidate *k*-subgraphs that are infrequent

- Details much more complicated, try to touch the main issues in the following

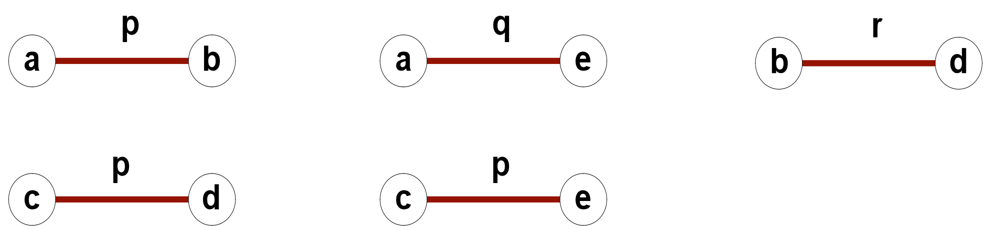
Example

Minimum support count = 2

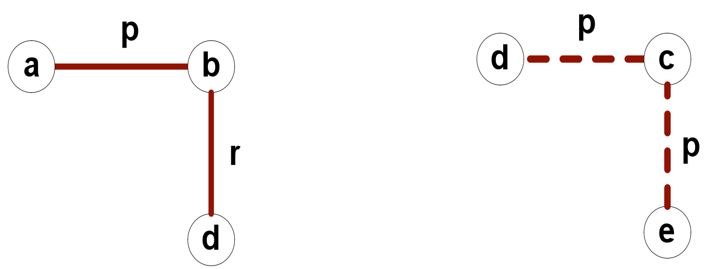
k=1
Frequent
Subgraphs



k=2
Frequent
Subgraphs



k=3
Candidate
Subgraphs



(Pruned candidate)

Candidate generation

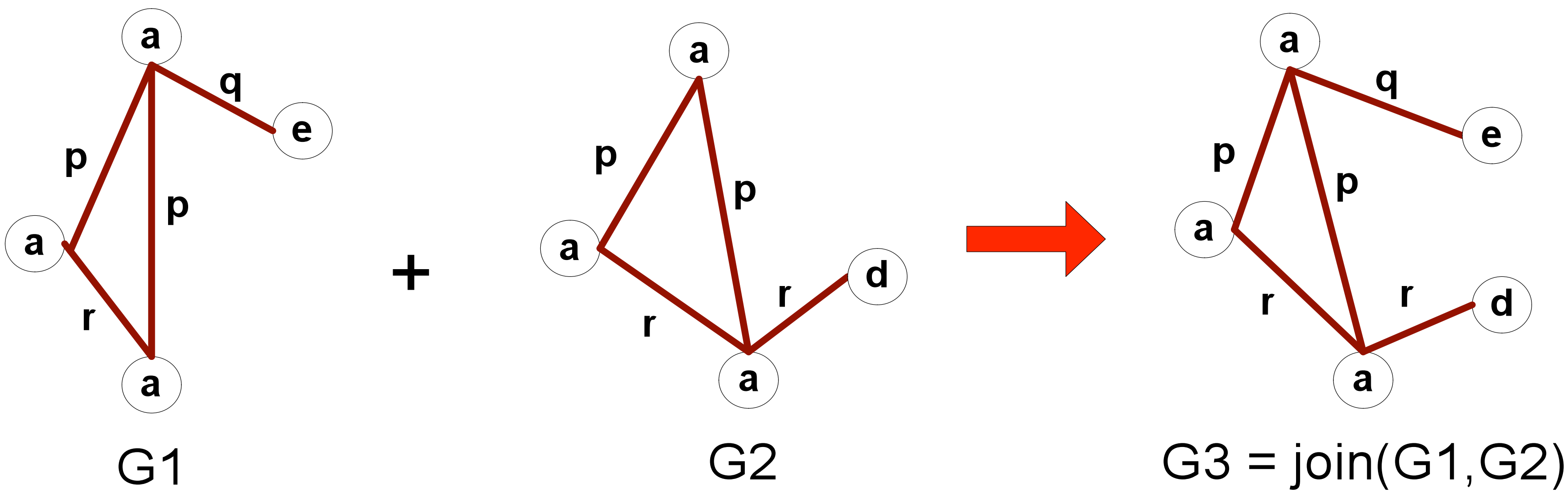
- Goal: is to merge a pair of k-1-subgraphs to create a k-subgraph
- First need to define k:
  - Number of vertices: merge two subgraphs that have k-1 vertices
  - Number of edges: merge two subgraphs that have k-1 edges
- How to avoid generating the same subgraph many times
  - Require that the k-1 subgraphs share a common k-2 subgraph, called a *core*

Candidate Generation: difference to itemset mining

- In Apriori:
  - Merging two frequent *k*-itemsets will produce a candidate (*k+1*)-itemset
- In frequent subgraph mining
  - Merging two frequent *k*-subgraphs will in general produce more than one candidate (*k+1*)-subgraph

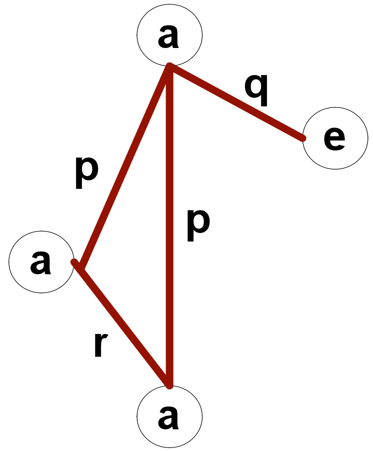
Candidate generation via vertex growing

- Generate a candidate by merging two subgraphs (G1, G2) that have a common core (subgraph of k-2 vertices) plus 1 extra vertex each
- A set of candidates will be generated that differ by one edge (d,e) and its label: G3 below is the candidate without the edge (d,e)



Adjacency matrix representation

- The vertex-growing approach can be viewed in terms of combining adjacency matrices of the subgraphs
- In our adjacency matrix representation
  - Rows and columns correspond to nodes
    - non-zero cells along a row (column) correspond to neighbors
  - Cells correspond to edges
    - cell contains edge label (or zero if no edge)

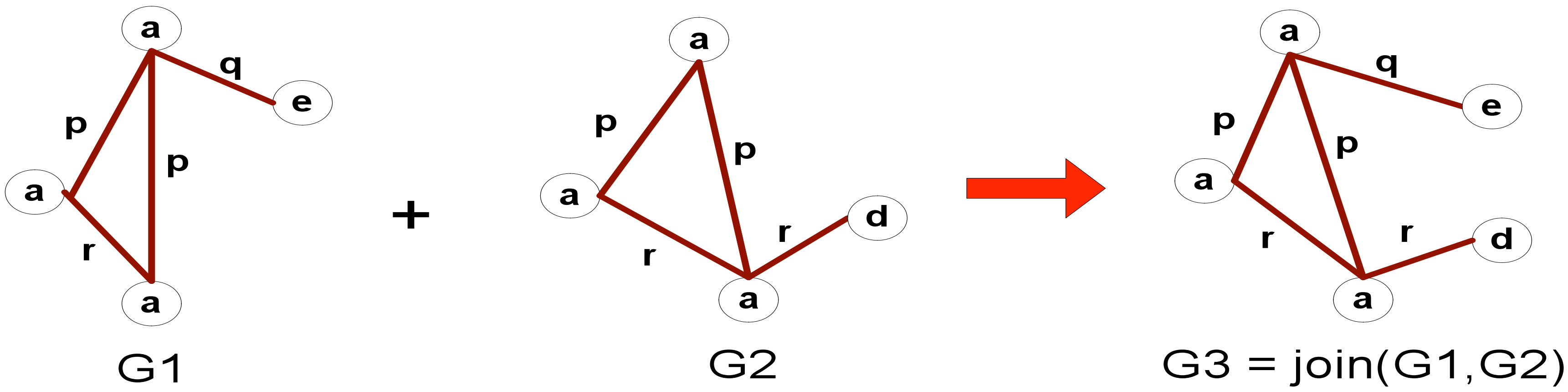


G1

$$M_{G1} = \begin{pmatrix} 0 & p & p & q \\ p & 0 & r & 0 \\ p & r & 0 & 0 \\ q & 0 & 0 & 0 \end{pmatrix}$$

Vertex Growing and adjacency matrices

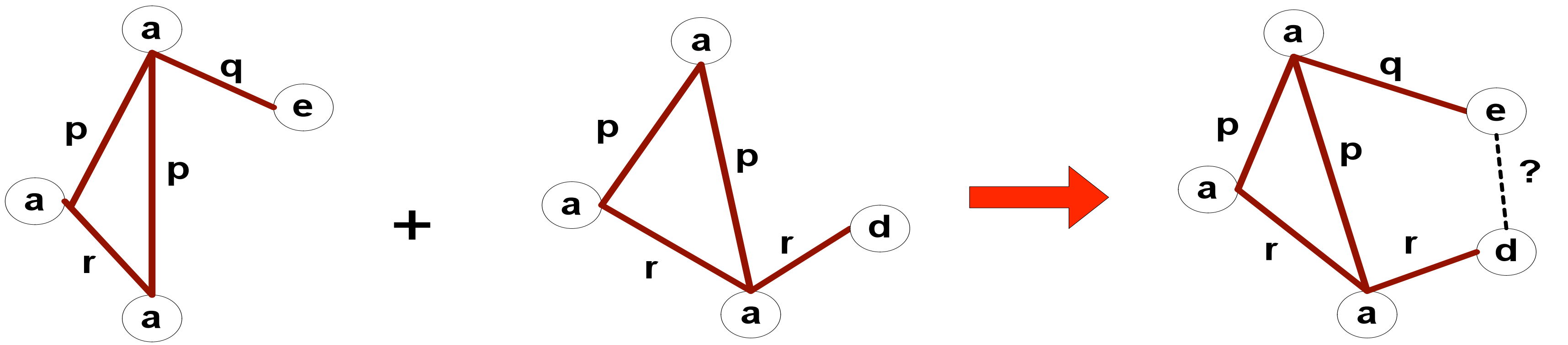
- Vertex growing takes two adjacency matrices that differ in the last row, and creates and augmented matrix by adding the last row and last column of the second matrix to the first matrix.



$$M_{G1} = \begin{pmatrix} 0 & p & p & q \\ p & 0 & r & 0 \\ p & r & 0 & 0 \\ q & 0 & 0 & 0 \end{pmatrix} \qquad M_{G2} = \begin{pmatrix} 0 & p & p & 0 \\ p & 0 & r & 0 \\ p & r & 0 & r \\ 0 & 0 & r & 0 \end{pmatrix} \qquad M_{G3} = \begin{pmatrix} 0 & p & p & 0 & q \\ p & 0 & r & 0 & 0 \\ p & r & 0 & r & 0 \\ 0 & 0 & r & 0 & 0 \\ q & 0 & 0 & 0 & 0 \end{pmatrix}$$

Multiplicity of Candidates in vertex growing

- A separate candidate is generated for each possible label of the edge (d,e)

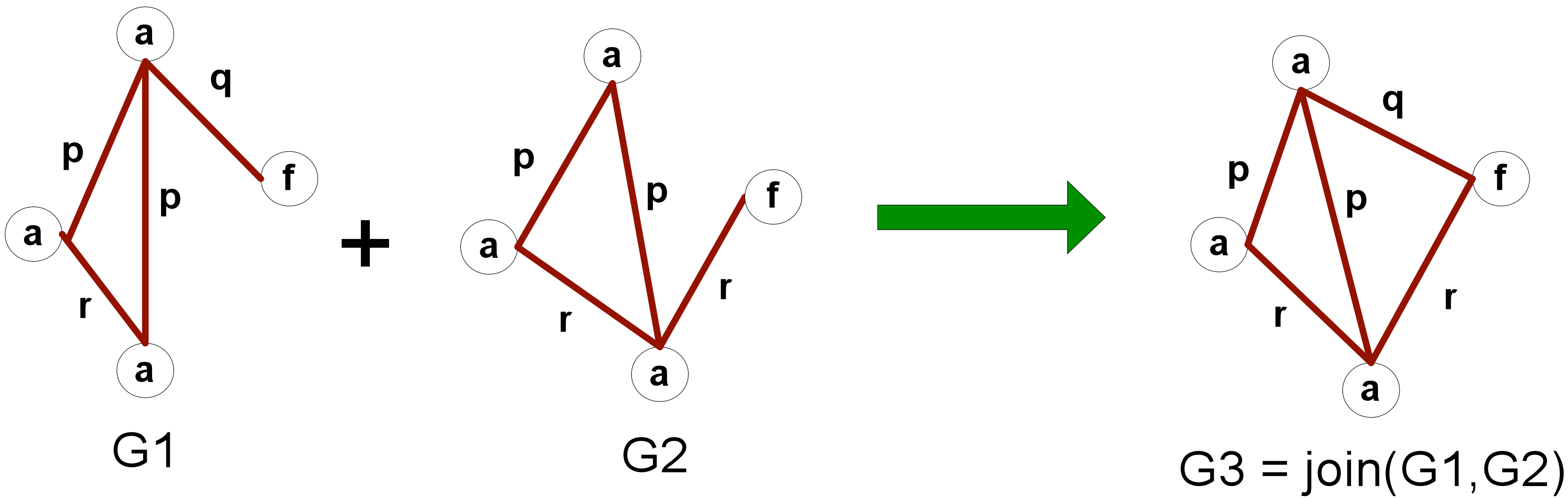


G1 G2 G3 = join(G1,G2)

$$M_{G1} = \begin{pmatrix} 0 & p & p & q \\ p & 0 & r & 0 \\ p & r & 0 & 0 \\ q & 0 & 0 & 0 \end{pmatrix} \qquad M_{G2} = \begin{pmatrix} 0 & p & p & 0 \\ p & 0 & r & 0 \\ p & r & 0 & r \\ 0 & 0 & r & 0 \end{pmatrix} \qquad M_{G3} = \begin{pmatrix} 0 & p & p & 0 & q \\ p & 0 & r & 0 & 0 \\ p & r & 0 & r & 0 \\ 0 & 0 & r & 0 & ? \\ q & 0 & 0 & ? & 0 \end{pmatrix}$$

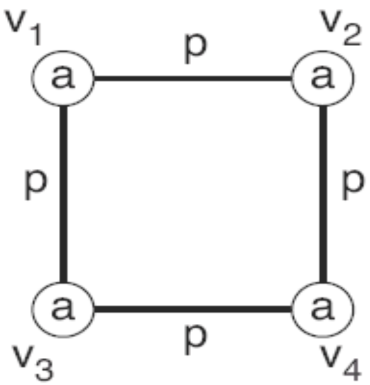
Candidate Generation via Edge Growing

- Edge growing approach inserts a new edge to an existing frequent subgraph
- Number of vertices not necessarily increased
- Criterion for merging is topological equivalence of the core

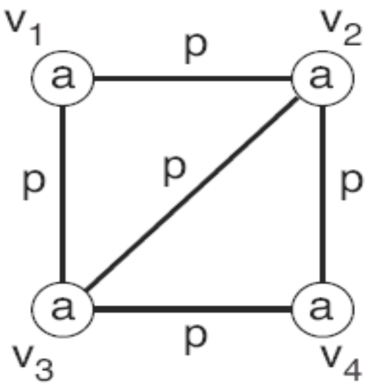


Topological equivalence

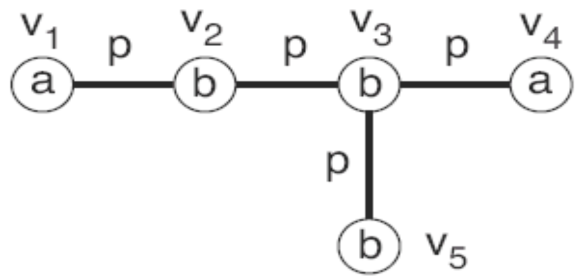
- All vertices in G1 are topologically equivalent: no matter where we add an edge, the resulting graph will have the same topology
- G2 contains two pairs of topologically equivalent vertices v1 & v4, v2 & v3: adding an edge to v1 or v4 will give one topology, adding an edge to v2 or v3 will give another topology
- G3 contains no topologically equivalent vertices: any choice of a vertex will lead to a different topology



G1



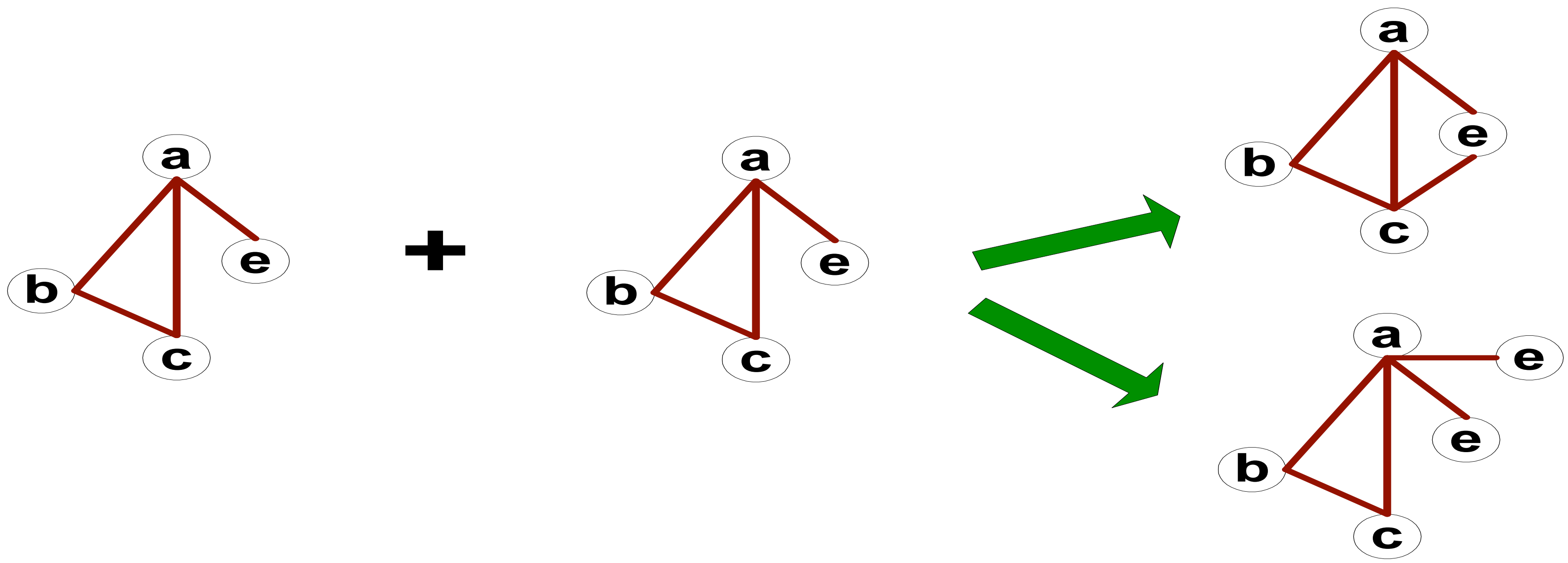
G2



G3

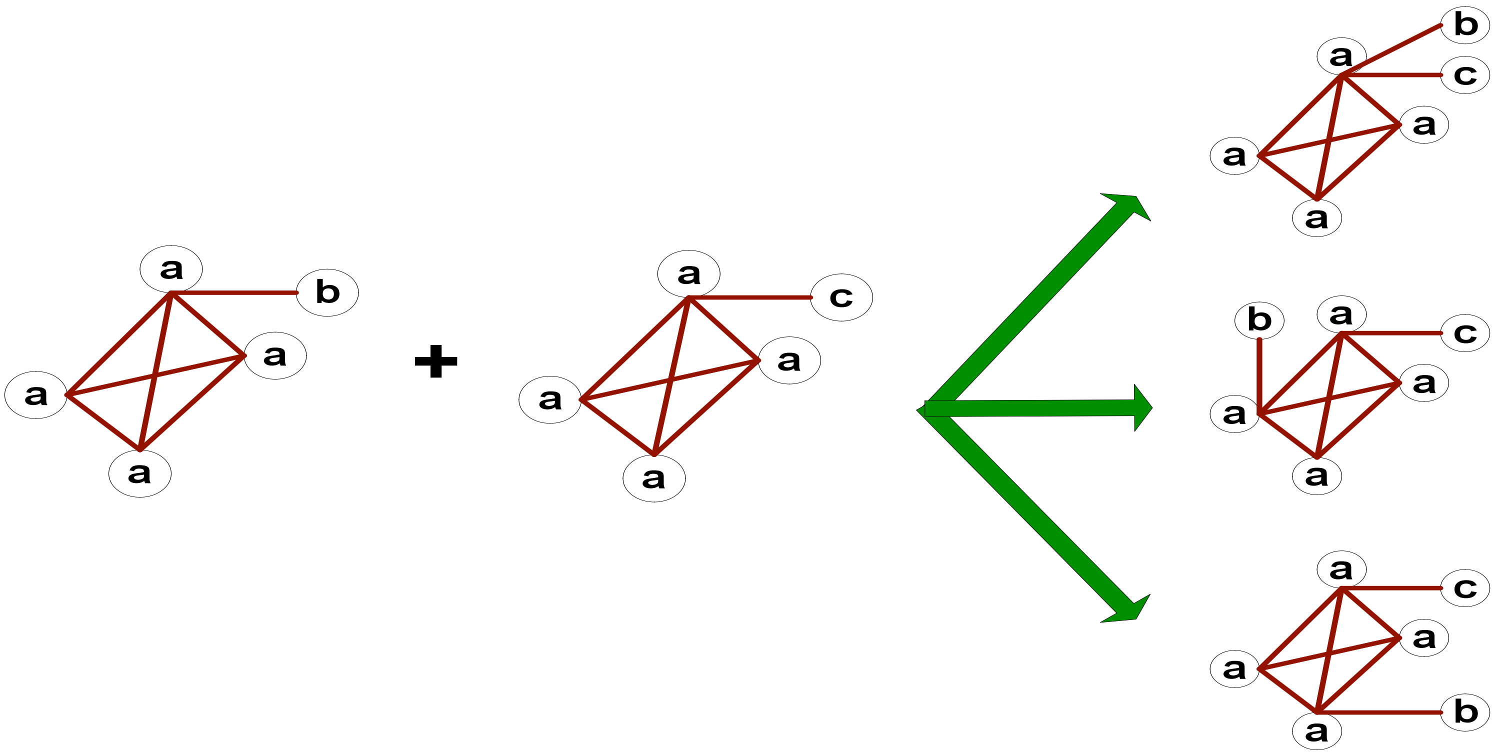
Multiplicity of Candidates in Edge growing

- Edge growing approach creates multiple candidates of three different kinds
- Case 1: topologically equivalent vertices (e) can be mapped to a single vertex or a pair of vertices



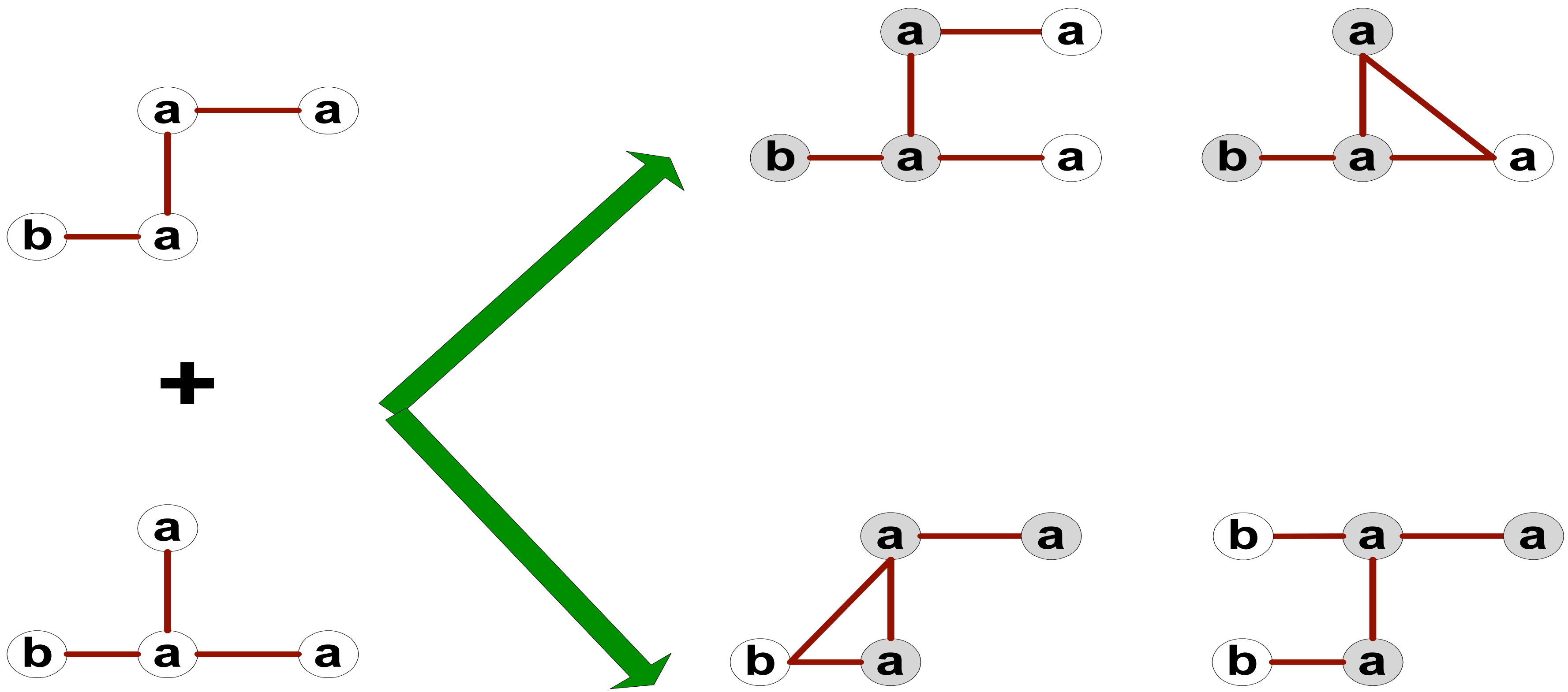
Multiplicity of Candidates in Edge growing

- Case 2: *Core* contains topologically equivalent vertices
- All symmetric orientations of the core generate potentially a different candidate



Multiplicity of Candidates in Edge growing

- Case 3: Core multiplicity
- Depending on how we select the core, we get different candidates



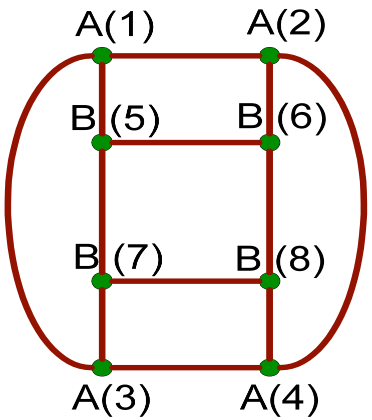
Candidate pruning

- Given a candidate k-subgraph, we need to check whether all the k-1-subgraphs are frequent
- Approach:
  - Successively remove one edge from the k-subgraph
  - If the resulting k-1 subgraph is connected check whether it is frequent
  - If not, remove the k-subgraph from the candidates
- Checking whether a k-1-subgraph is contained in the list of frequent k-1-subgraphs is not easy
  - Requires solving graph isomorphism problem, i.e. checking whether two graphs are topoplogically equivalent

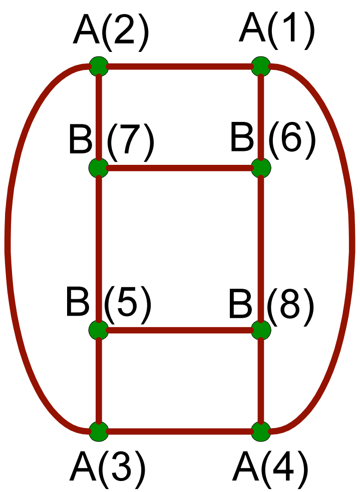
Hardness of graph matching

- Subgraph isomorphism:
  - Determining if a graph G contains another graph G’ as its subgraph is known as the subgraph isomorphism problem
  - One of the classical NP-hard problems, so no polynomial-time algorithm likely to exist
  - Needed in identification of the common core and support counting
- Graph isomorphism:
  - Determining if two graphs are topologically equivalent is the graph isomorphism problem
  - Complexity is not known, but no polynomial-time algorithm known
  - Needed: in candidate generation step, to determine whether a candidate has been generated and candidate pruning step, to check whether its (k-1)-subgraphs are frequent

Redundancy in the 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

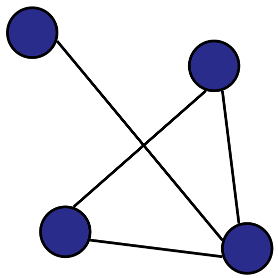


	A(1)	A(2)	A(3)	A(4)	B(5)	B(6)	B(7)	B(8)
A(1)	1	1	0	1	0	1	0	0
A(2)	1	1	1	0	0	0	1	0
A(3)	0	1	1	1	1	0	0	0
A(4)	1	0	1	1	0	0	0	1
B(5)	0	0	1	0	1	0	1	1
B(6)	1	0	0	0	0	1	1	1
B(7)	0	1	0	0	1	1	1	0
B(8)	0	0	0	1	1	1	0	1

- The same graph can be represented in many ways

Graph Isomorphism

- 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



$$\longrightarrow \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

String: 0010001111010110

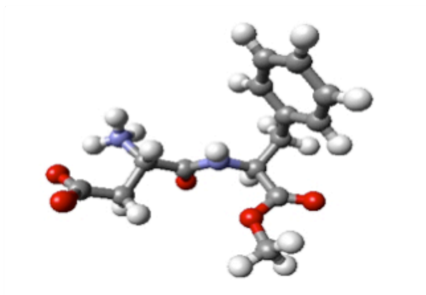
Canonical: 0111101011001000

Support counting

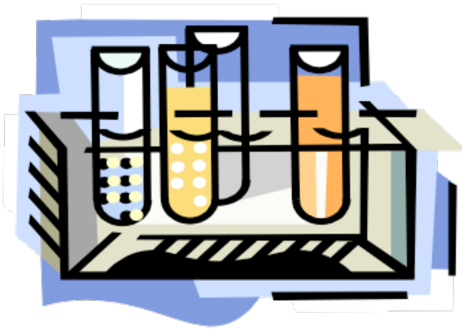
- Given a candidate k-subgraph, we need to check its support in out set of graphs
- Basic approach is to solve the subgraph isomorphism for each graph in the database
- More efficient is to
  - store the graphs that contain k-1 subgraphs in list of graph IDs (‘TID sets’)
  - intersect the lists of graph IDs of the k-1 subgraphs that generated the current graph, and
  - only compute the subgraph isomorphism between the k-subgraph and the graphs that are contained in the intersection

Graph mining application: Drug Discovery

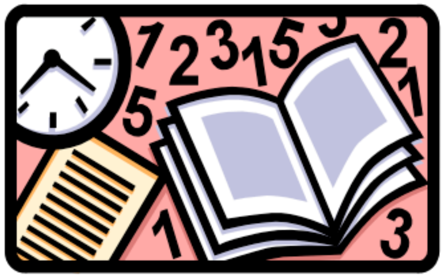
Idea for drug target



Drug screening/ rational drug design/ direct synthesis

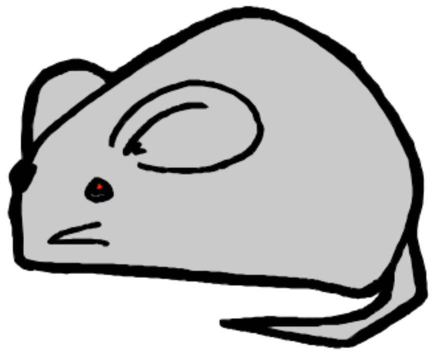


Small scale production



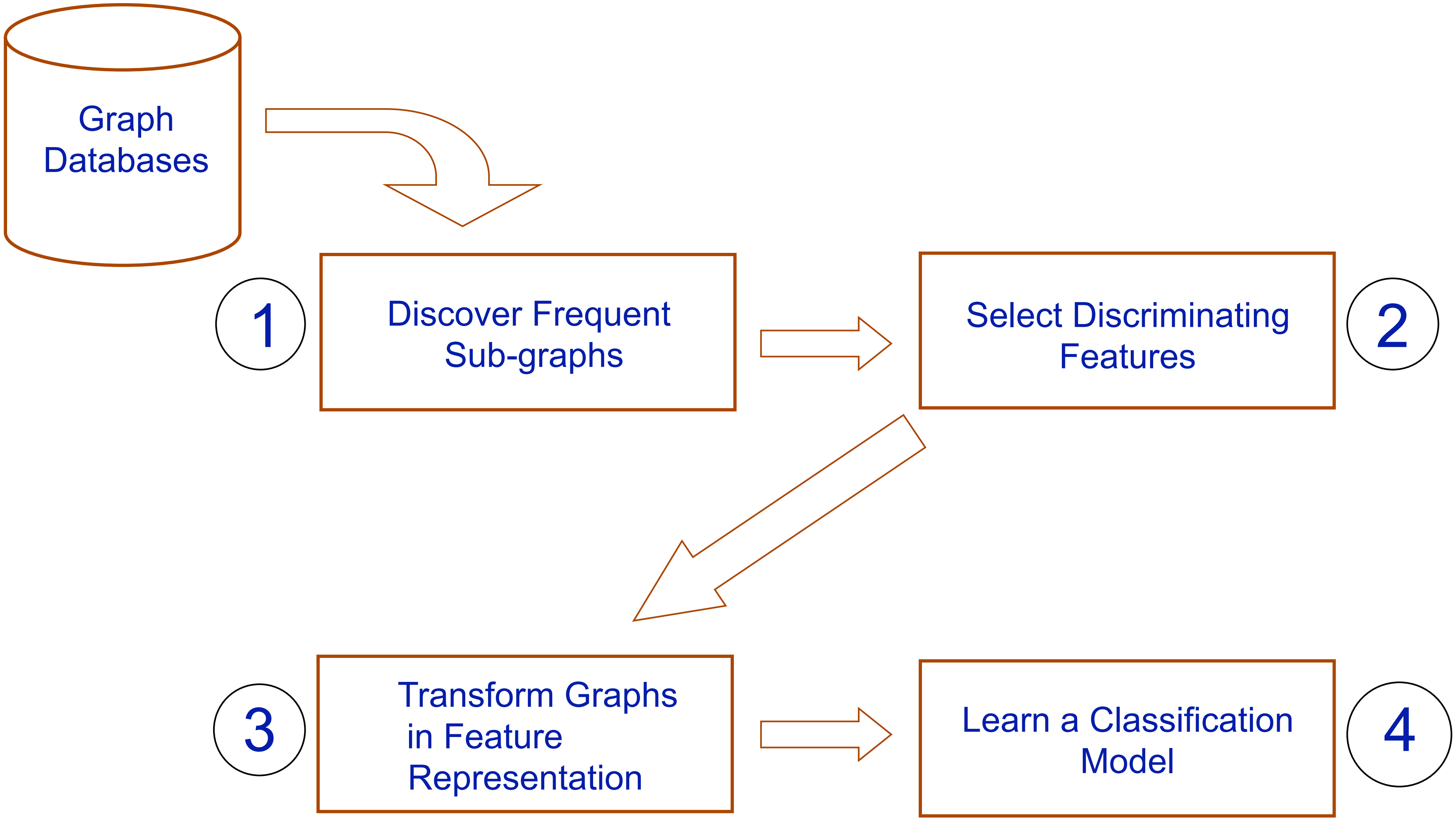
Investigational new drug

Production for clinical trials



Laboratory and animal testing

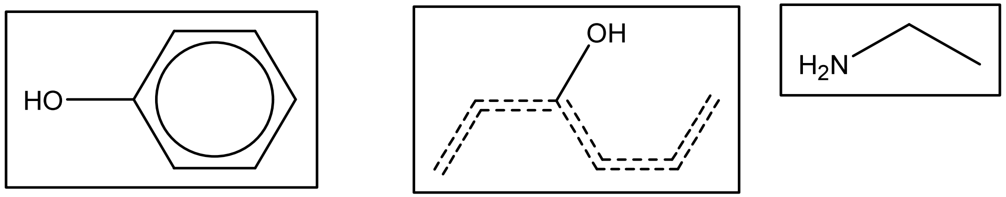
Graph Classification Approach



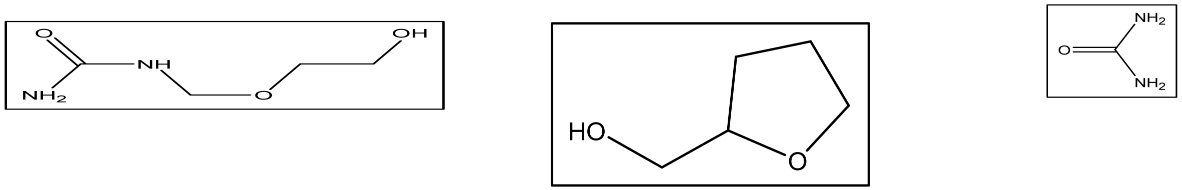
Chemical Compound Datasets

- Predictive Toxicology Challenge (PTC)
  - Predicting toxicity (carcinogenicity) of compounds.
  - Bio assays on four kinds of rodents
  - 4 Classification Problems -- Approx 400 chemical compounds.
- DTP AIDS Antiviral Screen (AIDS)
  - Predicting anti-HIV activity of compounds.
  - Assay to measure protection of human cells against HIV infection.
  - 3 Classification problems -- Approx 40,000 chemical compounds.
- Anthrax
  - Predicting binding ability of compounds with the anthrax toxin.
  - Expensive molecular dynamics simulation
  - Collaboration with Dr. Frank Lebeda, USAMRIID
  - Approx 35,000 chemical compounds

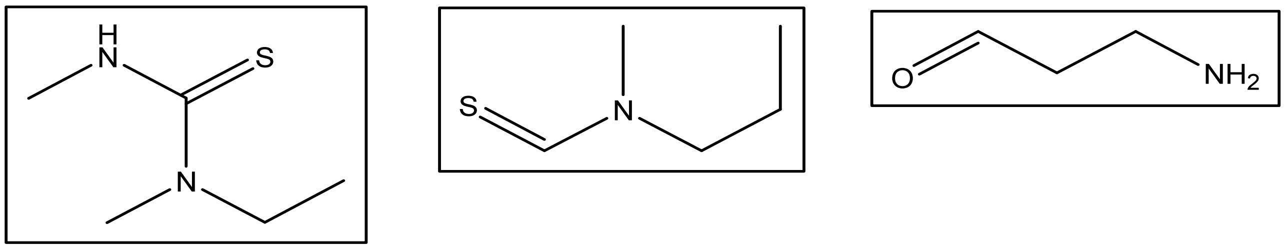
Most Discriminating Subbgraphs



(a) On Toxicology (PTC) Dataset



(b) On AIDS Dataset



(c) On Anthrax Dataset

582671 Graph Mining - Motivation, Algorithms and Applications (2 cp)

- Special course during the intensive period
- Preliminary dates: 18-27 May 2010
- Teacher: Professor Ehud Gudes, Ben-Gurion University of the Negev Beer-Sheva, Israel
- Coordinator: Greger Lindén
- Place: Department of Computer Science, Exactum Building. Gustaf Hällströmin katu 2b
- Enroll at http://ilmo.cs.helsinki.fi/