Complex Data Mining & Workflow Mining

Graph Mining and Workflow Mining

ICAR CNR
Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Outline

- Introduzione e concetti di base
  - Motivazioni, applicazioni
  - Concetti di base nell'analisi dei dati complessi
- Web/Text Mining
  - Concetti di base sul Text Mining
  - Tecniche di data mining su dati testuali
- Graph Mining
  - Introduzione alla graph theory
  - Principali tecniche e applicazioni
- Workflow Mining
  - I workflow: grafi con vincoli
  - Frequent pattern discovery su workflow: motivazioni, metodi, applicazioni
- Multi-Relational data mining
  - Motivazioni: da singole tabelle a strutture complesse
  - Alcune delle tecniche principali

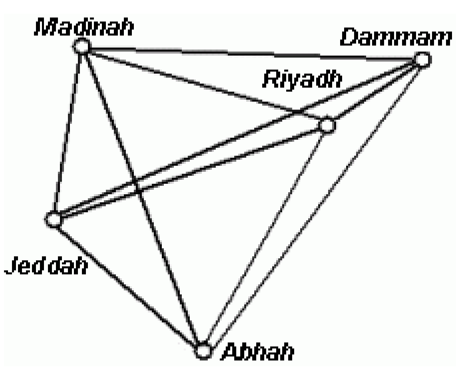
Introduction to Graphs

- What is a Graph?
- Some example applications of graphs.
- Graph Terminology.
- Representation of Graphs.
  - Adjacency Matrix.
  - Adjacency Lists.
  - Simple Lists
- Review Questions.

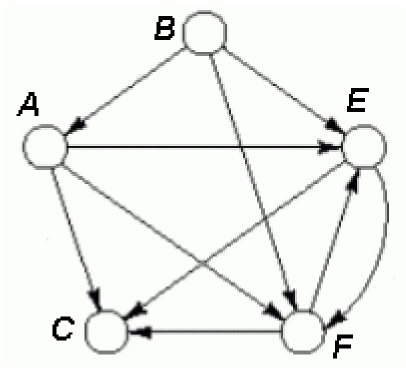
What is a Graph?

- Graphs are Generalization of Trees.
- A simple graph G = (V, E) consists of a non-empty set V, whose members are called the vertices of G, and a set E of pairs of distinct vertices from V, called the edges of G.

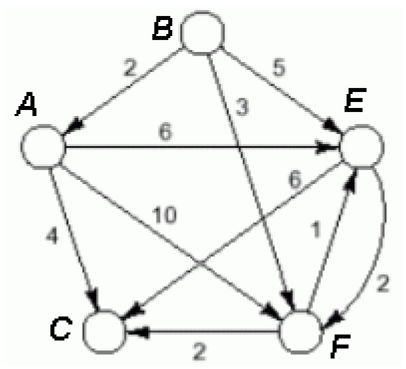
Undirected



Directed (Digraph)



Weighted



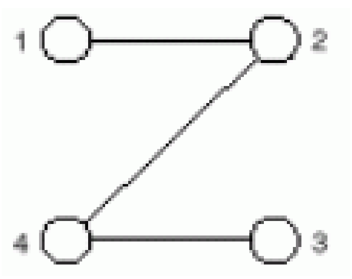
Some Example Applications of Graph

- Finding the least congested route between two phones, given connections between switching stations.
- Determining if there is a way to get from one page to another, just by following links.
- Finding the shortest path from one city to another.
- As a traveling sales-man, finding the cheapest path that passes through all the cities the sales-man must visit
- Determining an ordering of courses so that prerequisite courses are always taken first.

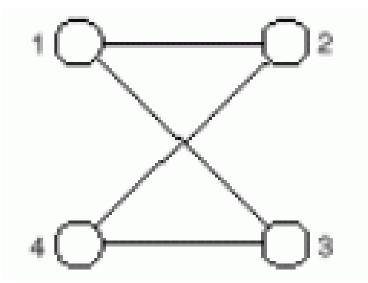
Graphs Terminology

- Adjacent Vertices: there is a connecting edge.
- Path: A sequence of adjacent vertices.
- Cycle: A path in which the last and first vertices are adjacent.
- Connected graph: There is a path from any vertex to every other vertex.

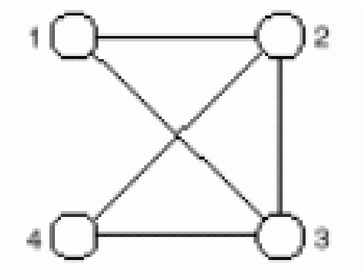
Path



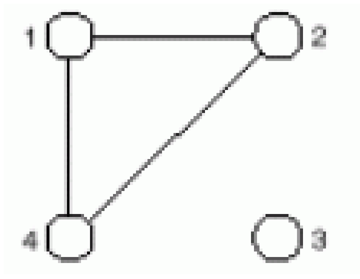
Cycle



Connected



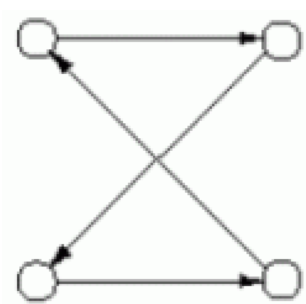
Disconnected



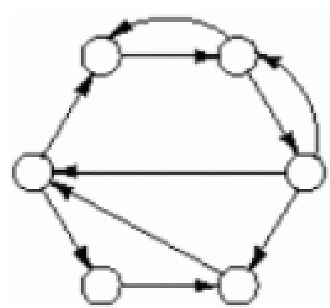
More Graph Terminologies

- Path and cycles in a digraph: must move in the direction specified by the arrow.
- Connectedness in a digraph: strong and weak.
- Strongly Connected: If connected as a digraph - following the arrows.
- Weakly connected: If the underlying undirected graph is connected (i.e. ignoring the arrows).

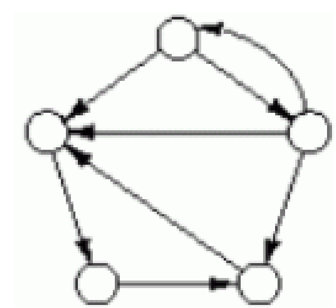
Directed Cycle



Strongly Connected



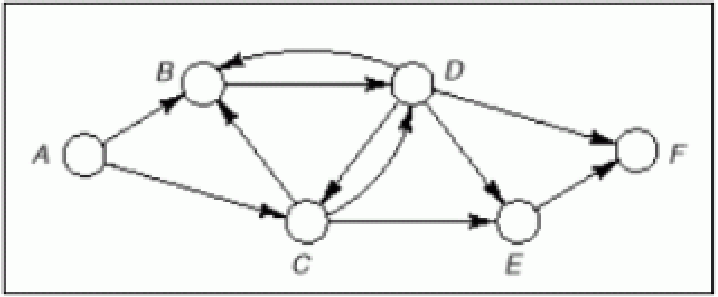
Weakly Connected



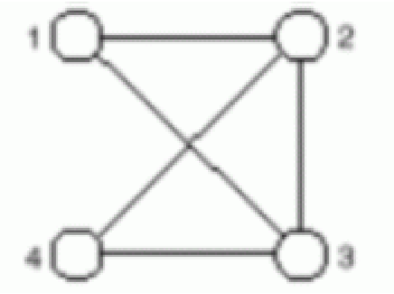
Further Graph Terminologies

- Emanate: an edge e = (v, w) is said to emanate from v.
  - A(v) denotes the set of all edges emanating from v.
- Incident: an edge e = (v, w) is said to be incident to w.
  - I(w) denote the set of all edges incident to w.
- Out-degree: number of edges emanating from v -- |A(v)|
- In-degree: number of edges incident to w -- |I(w)|.

Directed Graph



Undirected Graph



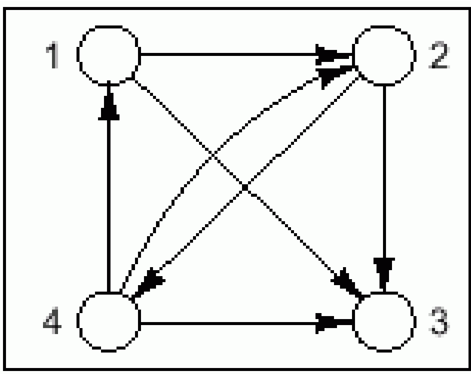
Graph Representations

- For vertices:
  - an array or a linked list can be used

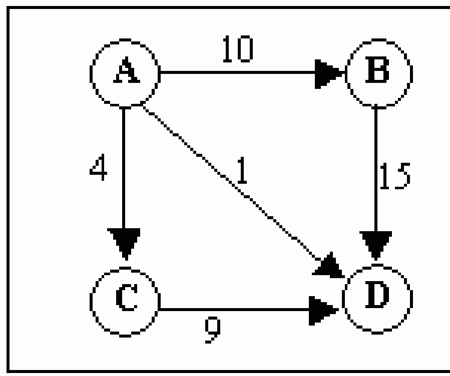
- For edges:
  - Adjacency Matrix (Two-dimensional array)
  - Adjacency List (One-dimensional array of linked lists)
  - Linked List (one list only)

Adjacency Matrix Representation

- Uses a 2-D array of dimension |V|x|V| for edges
- The presence/absence of the edge *(v, w)* is indicated by the entry in row *v* and column *w* of the matrix
- For an unweighted graph, boolean values could be used.
  - For a weighted graph, the actual weights are used.



	1	2	3	4
1	F	T	T	F
2	F	F	T	T
3	F	F	F	F
4	T	T	T	F

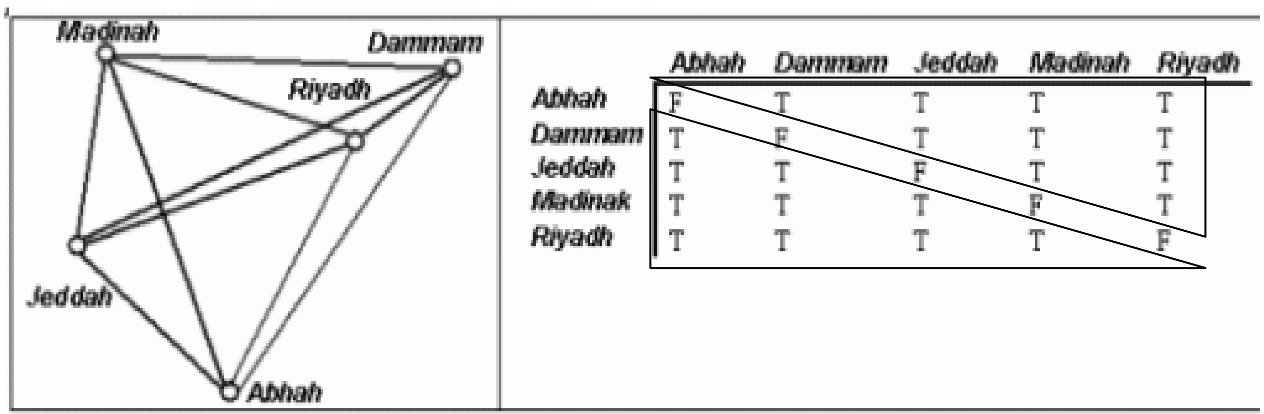


	A	B	C	D
A	∞	10	4	1
B	∞	∞	∞	15
C	∞	∞	∞	9
D	∞	∞	∞	∞

ICAR CNR
Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

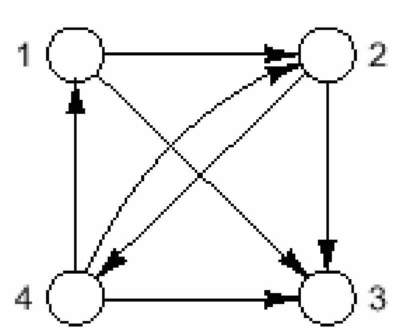
Notes on Adjacency Matrix

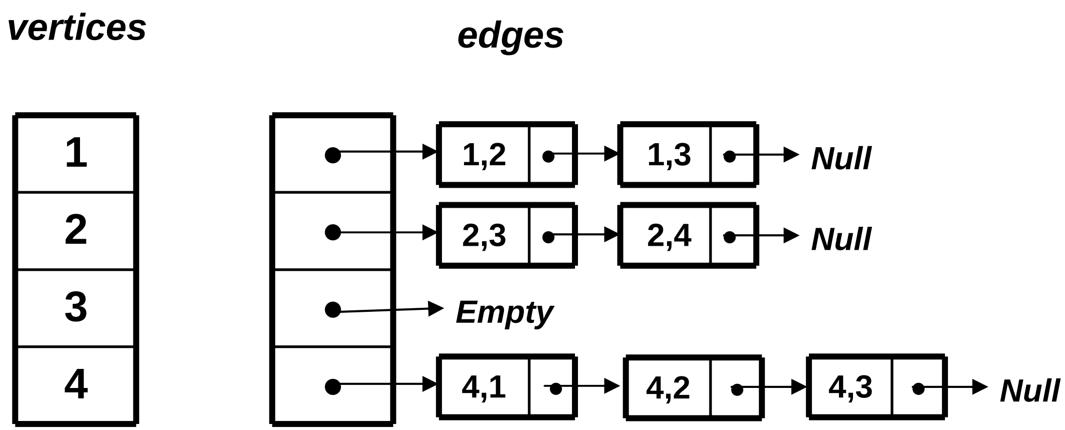
- For undirected graph, the adjacency matrix is always symmetric.
- In a Simple Graph, all diagonal elements are zero (i.e. no edge from a vertex to itself).
- The space requirement of adjacency matrix is O(n2) - most of it wasted for a graph with few edges.
  - However, entries in the matrix can be accessed directly.



Adjacency List Representation

- This involves representing the set of vertices adjacent to each vertex as a list. Thus, generating a set of lists.
- This can be implemented in different ways.
- Our representation:
  - Vertices as a one dimensional array
  - Edges as an array of linked list (the emanating edges of vertex 1 will be in the list of the first element, and so on, …

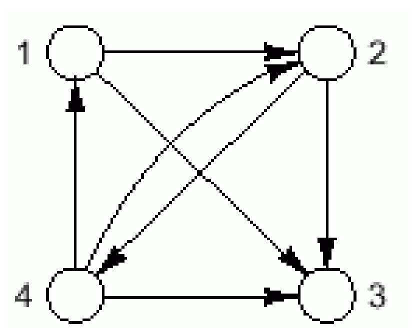




Simple List Representation

- Vertices are represented as a 1-D array or a linked list

- Edges are represented as one linked list
  - Each edge contains the information about its two vertices



vertices
1
2
3
4

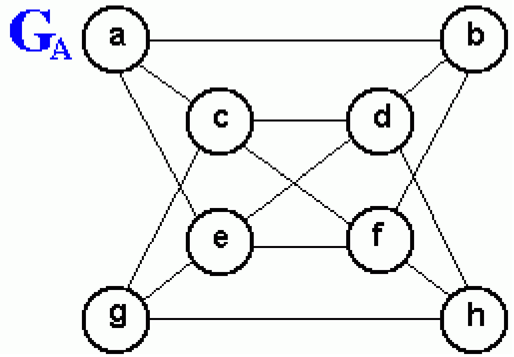
.
.
.

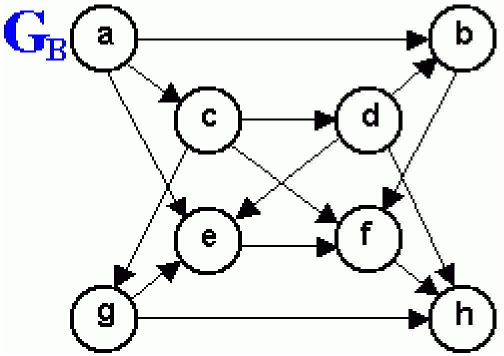
edges
edge(1,2)
edge(1,3)
edge(2,3)
edge(2,4)
edge(4,1)
edge(4,2)
edge(4,3)

.
.
.

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Questions





1) Consider the undirected graph GA shown above. List the elements of *V* and *E*. Then, for each vertex v in V, do the following:

- **Compute the in-degree of v**
- **Compute the out-degree of v**
- **List the elements of A(v)**
- **List the elements of I(v).**

2) Consider the undirected graph G_A shown above.

- **Show how the graph is represented using adjacency matrix.**
- **Show how the graph is represented using adjacency lists.**

3) Repeat Exercises 1 and 2 for the directed graph G_B shown above.

Two flavors of graph mining

- Mining over a single graph
  - Link analysis
  - Clustering nodes/arcs
- Mining a collection of graphs (graph dataset)
  - Frequent graph (sub-)patterns
  - Clustering similar graphs
  - Classification of a new graph

Mining over a single graph

Link analysis for the Web

- *Authorities* are pages that are recognized as providing significant, trustworthy, and useful information on a topic.
- *Hubs* are index pages that provide lots of useful links to relevant content pages (topic authorities).
- Algorithms
  - HITS
  - PageRank

Another link analysis application: Citation Analysis (Bibliometrics)

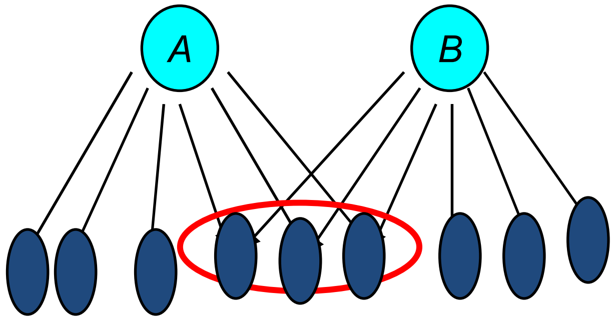
- Many standard documents include *bibliographies* (or *references*), explicit *citations* to other previously published documents.
- Using citations as links, standard corpora can be viewed as a graph.
- The structure of this graph, independent of content, can provide interesting information about the similarity of documents and the structure of information.

Impact Factor

- Developed by Garfield in 1972 to measure the importance (quality, influence) of scientific journals.
- Measure of how often papers in the journal are cited by other scientists.
- Computed and published annually by the Institute for Scientific Information (ISI).
- The *impact factor* of a journal *J* in year *Y* is the average number of citations (from indexed documents published in year *Y*) to a paper published in *J* in year *Y*–1 or *Y*–2.
- Does not account for the quality of the citing article.

Bibliographic Coupling

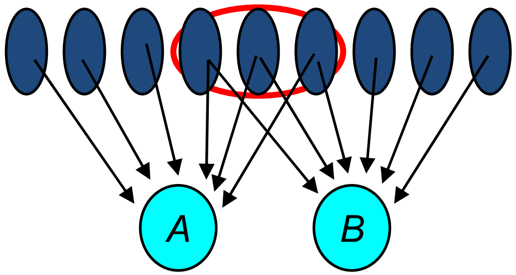
- Measure of similarity of documents introduced by Kessler in 1963.
- The bibliographic coupling of two documents *A* and *B* is the number of documents cited by *both* *A* and *B*.
- Size of the intersection of their bibliographies.
- Maybe want to normalize by size of bibliographies?



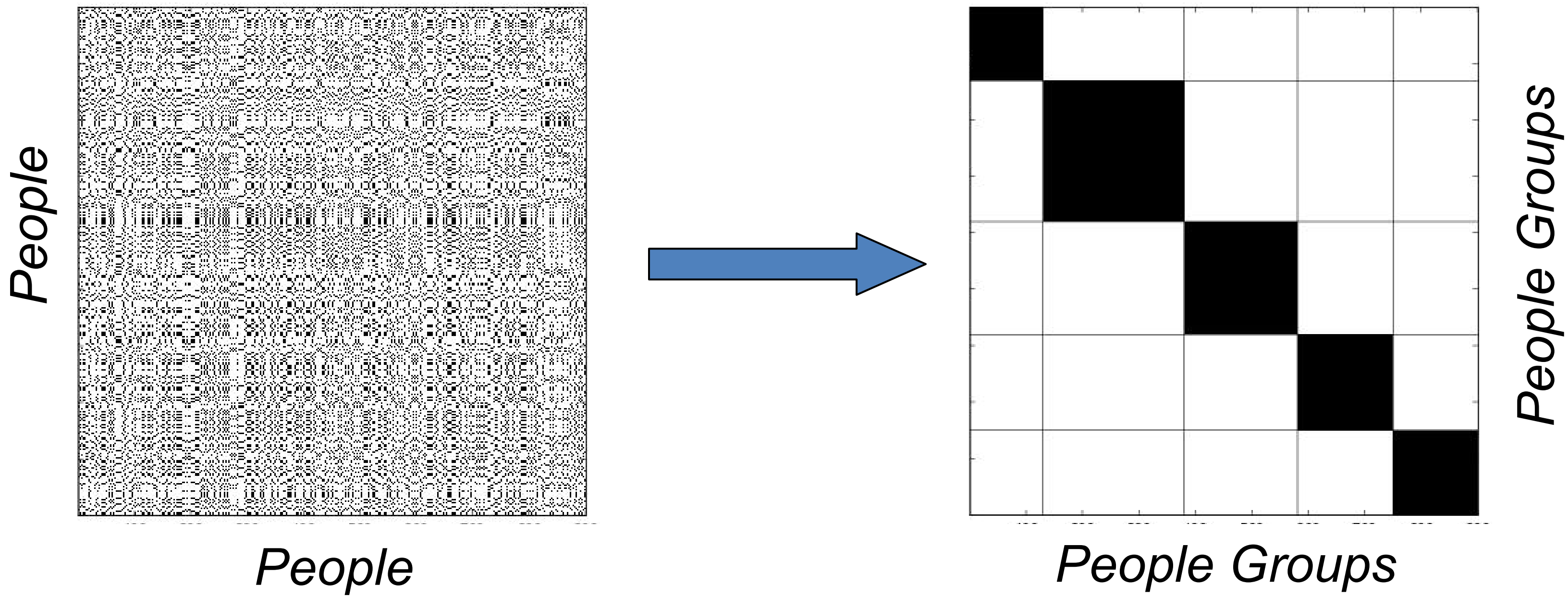
Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Co-Citation

- An alternate citation-based measure of similarity introduced by Small in 1973.
- Number of documents that cite both *A* and *B*.
- Maybe want to normalize by total number of documents citing either *A* or *B* ?



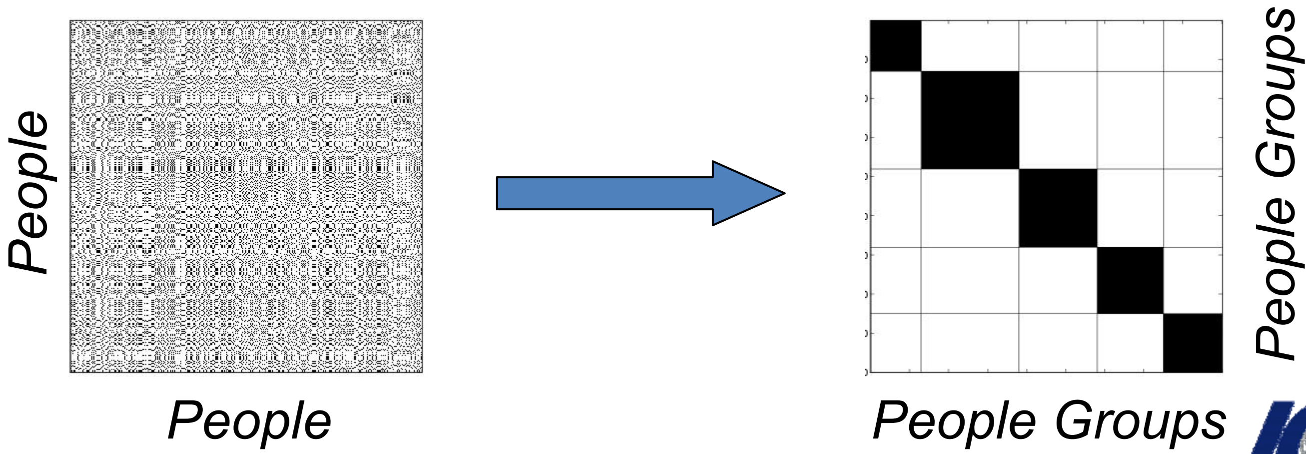
Clustering large graphs



- Group people in a social network,
  or, species in a food web,
  or, proteins in protein interaction graphs …

Clustering the adjacency matrix

- Find groups (of people, species, proteins, pages, etc.)
- Find outlier edges ("bridges")
- Compute inter-group distances ("how similar are two groups of pages?")



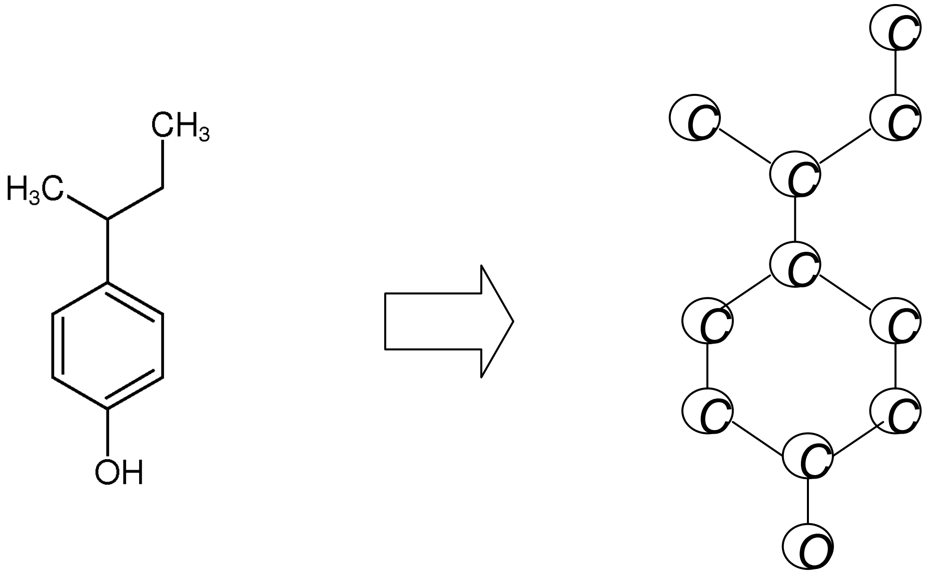
Approaches

- Graph Partitioning
  - Direct search for (quasi-)connected components
- Clustering Techniques
  - K-means and variants
    - Basically, post-process clusters to identify communities
  - Co-clustering
- LSI
  - Use matrix decomposition techniques

Mining from a collection of graphs

- Collections of graphs are ubiquitous
  - Chemical compounds (Cheminformatics)
  - Protein structures, biological pathways/networks (Bioinformactics)
  - Program control flow, traffic flow, and workflow analysis
  - XML databases, Web, and social network analysis
- Graphs are a general model
  - Trees, lattices, sequences, and items are degenerated graphs

Example: Molecule as a labeled graph



Main directions for mining graph datasets

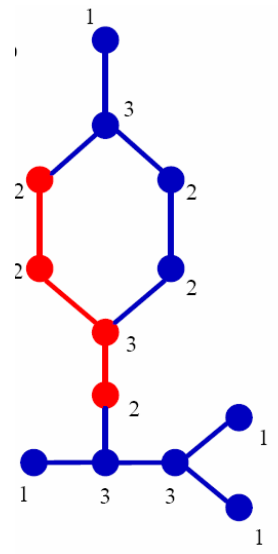
- Frequent graph (sub-)patterns
- Clustering similar graphs
- Classification of a graph

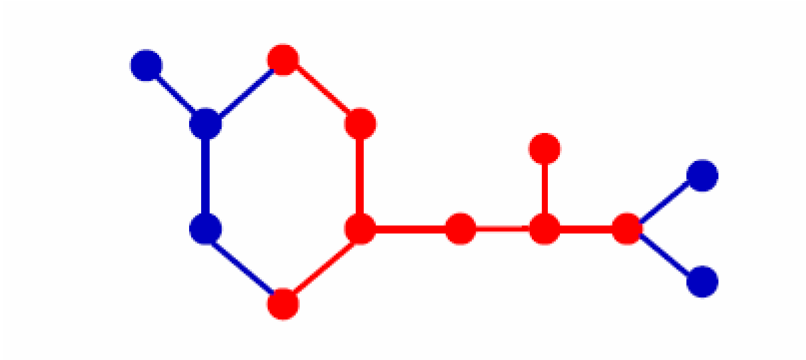
Graph Pattern Mining

- *Frequent* subgraphs
  - A (sub)graph is ***frequent*** if its *support* (occurrence frequency) in a given dataset is above a *minimum support* threshold
- Applications of graph pattern mining
  - Mining biochemical structures
  - Program control flow analysis
  - Workflow analysis (more to come next)
  - Mining XML structures or Web communities
  - Building blocks for graph classification, clustering, compression, comparison, and correlation analysis

Recall from Graph theory

- Path
  - A sequence of edges connecting two nodes
- Subgraph
  - A subset of nodes/edges

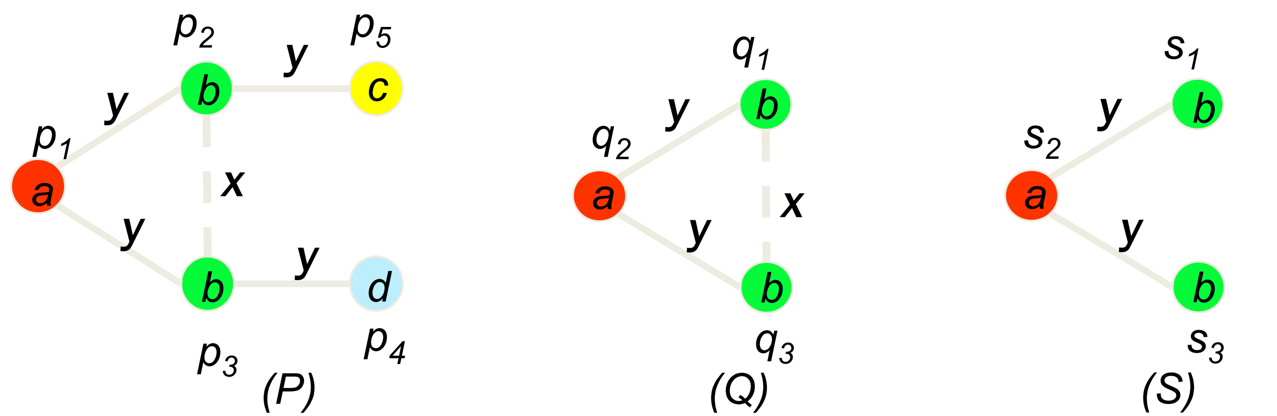




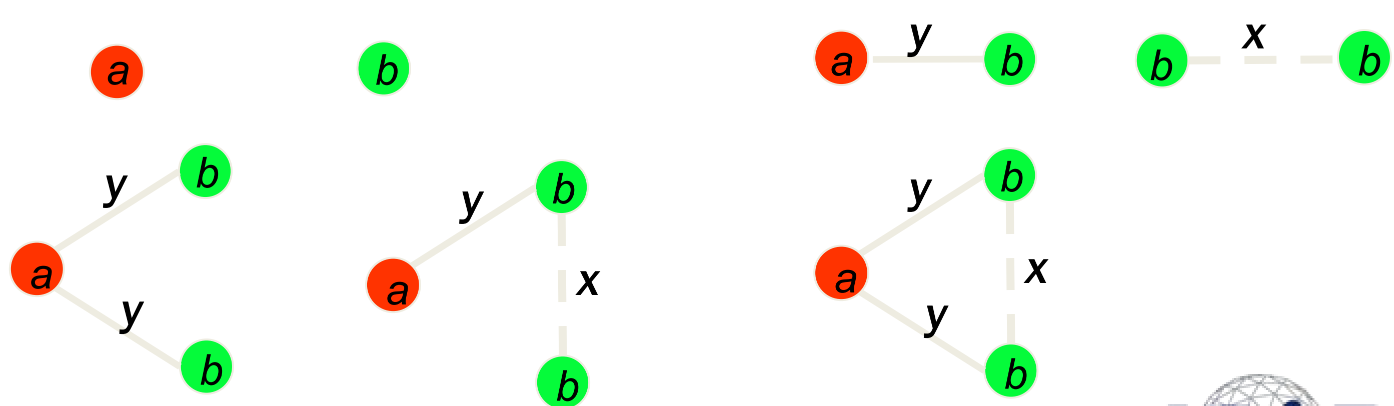
Frequent Subgraph Mining

Input: A set GD of labeled (un)directed graphs

σ = 2/3

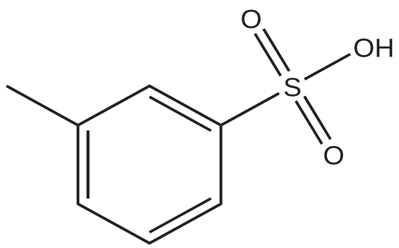


Output: All frequent subgraphs (w. r. t. σ) from GD.

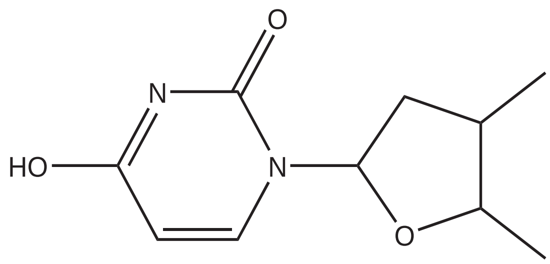


Example: Frequent Subgraphs

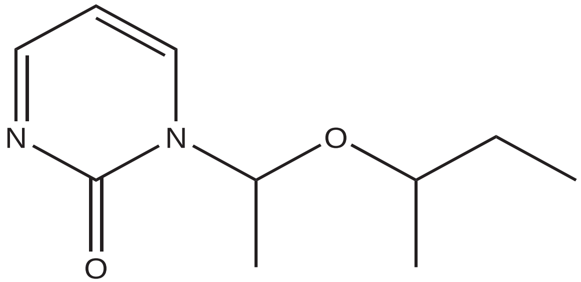
GRAPH DATASET



(A)



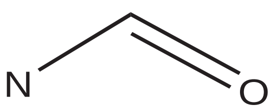
(B)



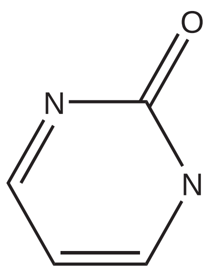
(C)

FREQUENT PATTERNS (MIN SUPPORT IS 2)

(1)

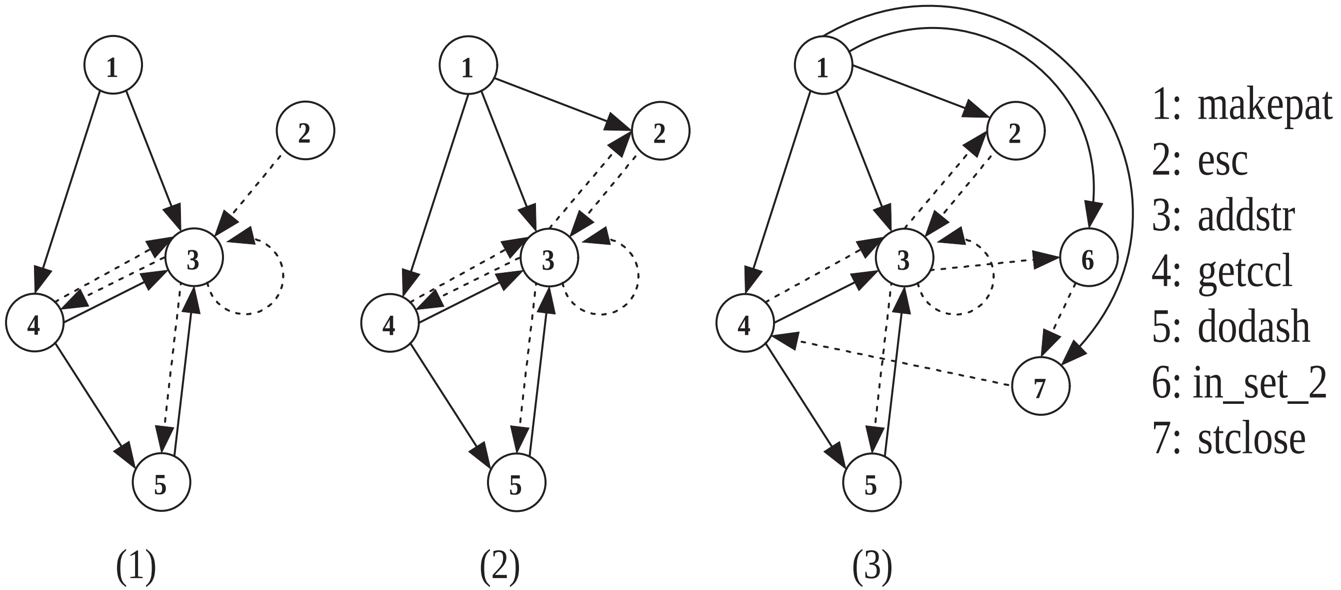


(2)

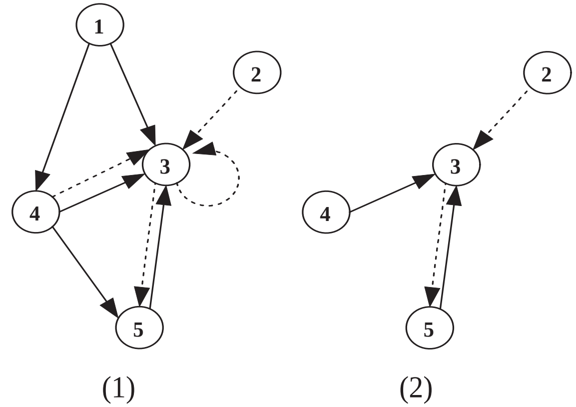


Example (II)

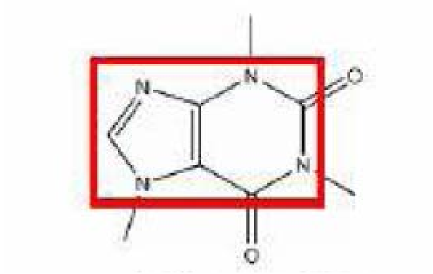
GRAPH DATASET



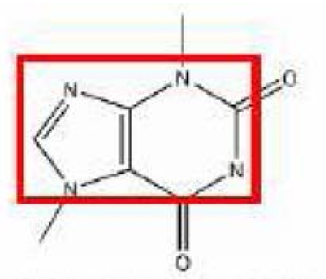
FREQUENT PATTERNS (MIN SUPPORT IS 2)



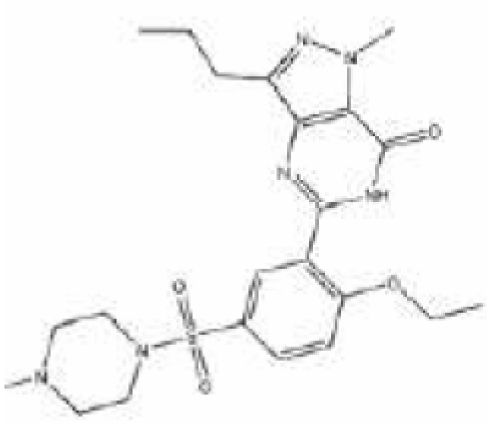
Example (III)



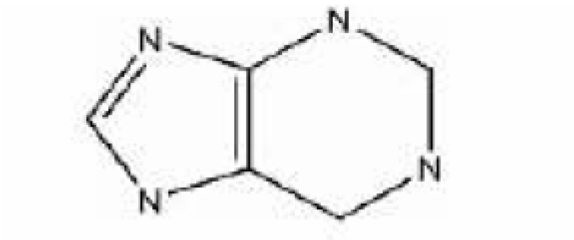
Caffeine



diurobromine



Viagra



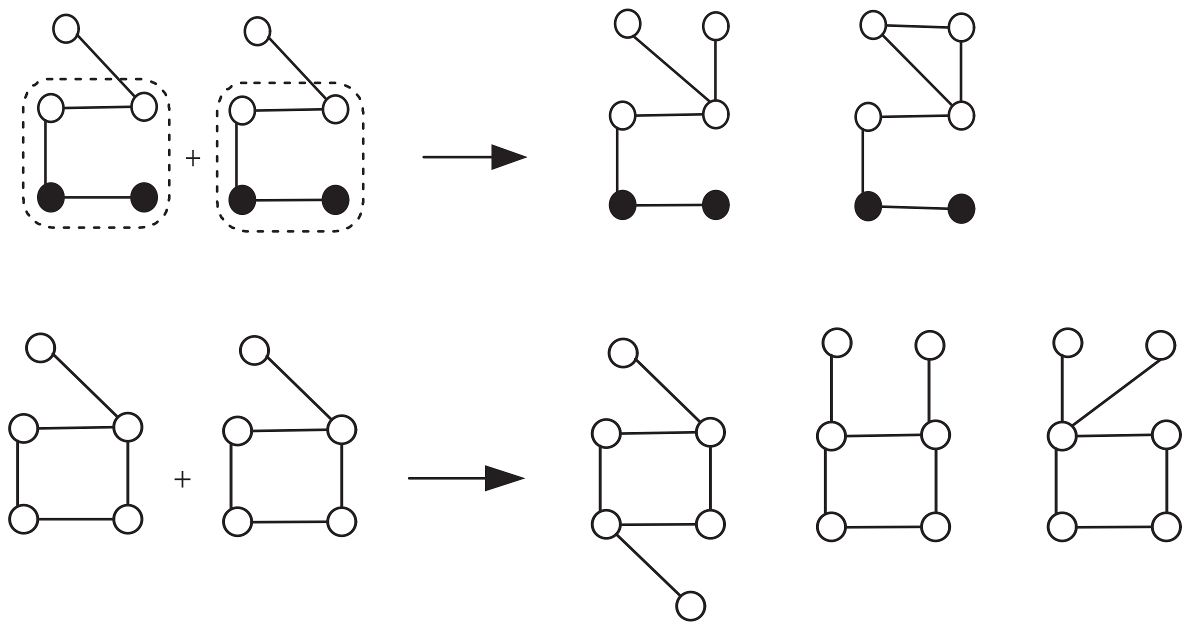
Frequent subgraph

Properties of Graph Mining Algorithms

- Search order
  - Breadth vs. depth
- Generation of candidate subgraphs
  - apriori vs. pattern growth
- Elimination of duplicate subgraphs
  - passive vs. active

Apriori-Based, Breadth-First Search

- Methodology: breadth-search
  - new graphs contain one more node, or one more edge only

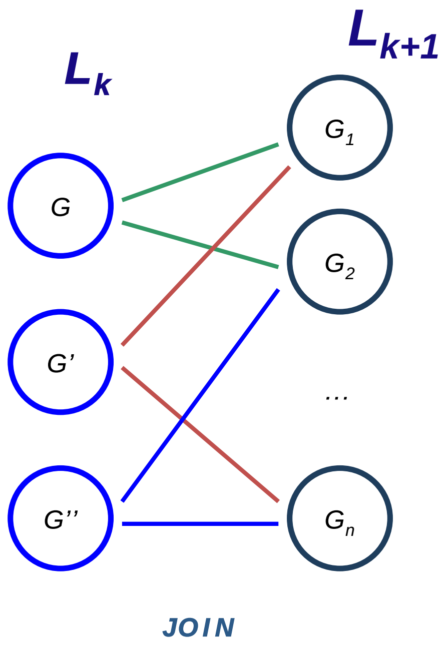


Apriori basic algorithm

- Generate candidates (subgraphs) in a level-wise manner
- Test that each candidate has not been already investigated
  - Isomorphism against a set of candidates
- Test that each candidate is a real subgraph of the data and take its frequency
  - Subgraph matching against the set of graphs

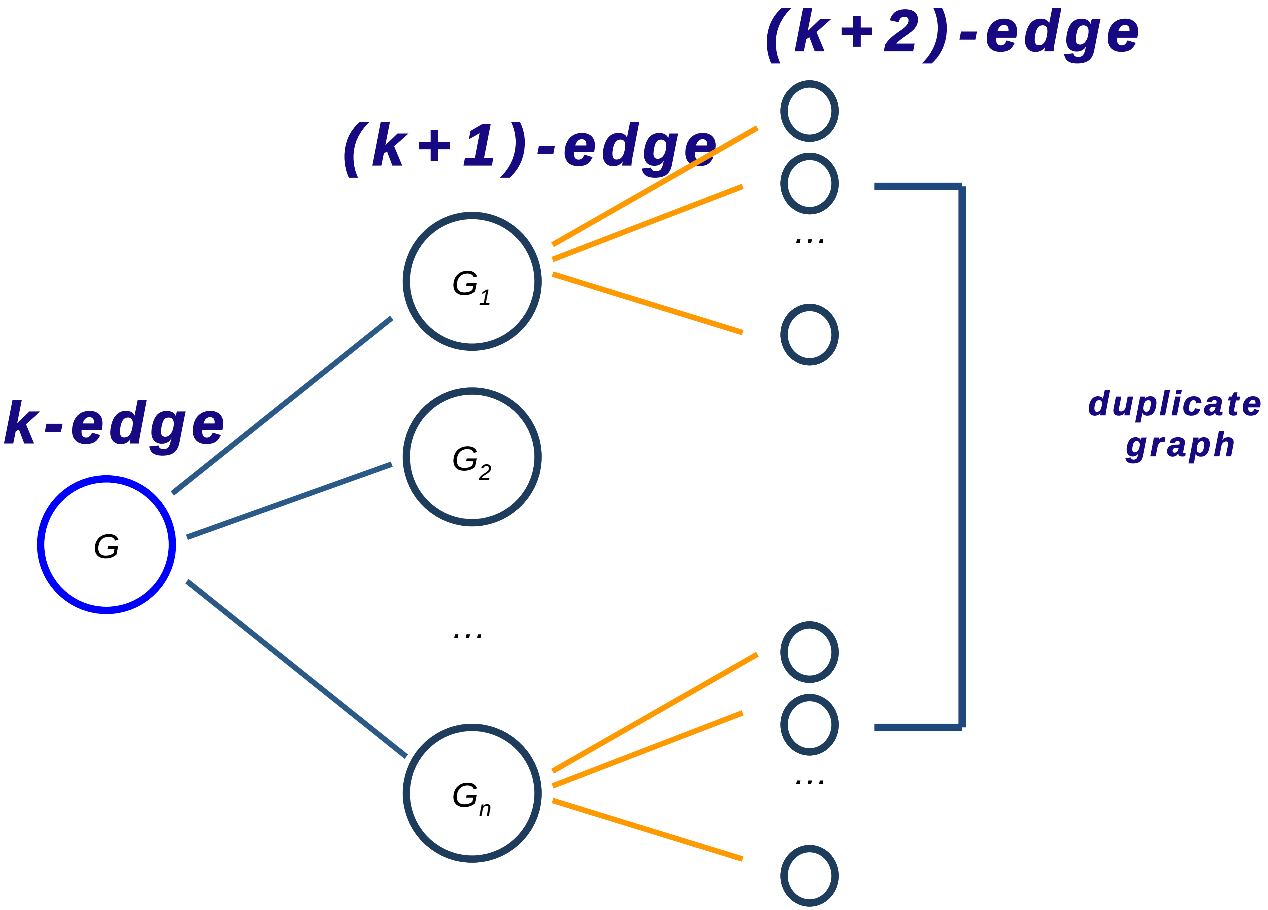
Apriori-based algorithm

1. L_1 = *{nodi frequenti};*
2. ***for*** *(k = 1;* $L_k \neq \varnothing$*; k++)* ***do begin***
3. $\quad C_{k+1}$ *= candidati generati da* L_k*;*
4. $\quad$ ***for each*** *graph g in D*
5. $\qquad$ *Incrementa il supporto dei candidati in* C_{k+1} *che sono sottografi di g;*
6. $\quad L_{k+1}$ *= tutti i candidati in* C_{k+1} *con min_support*
7. ***return*** $\cup_k L_k$*;*



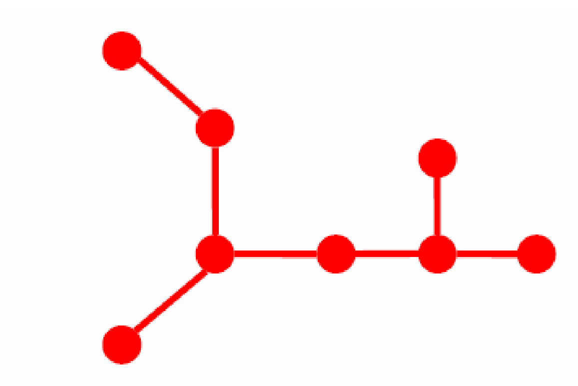
Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

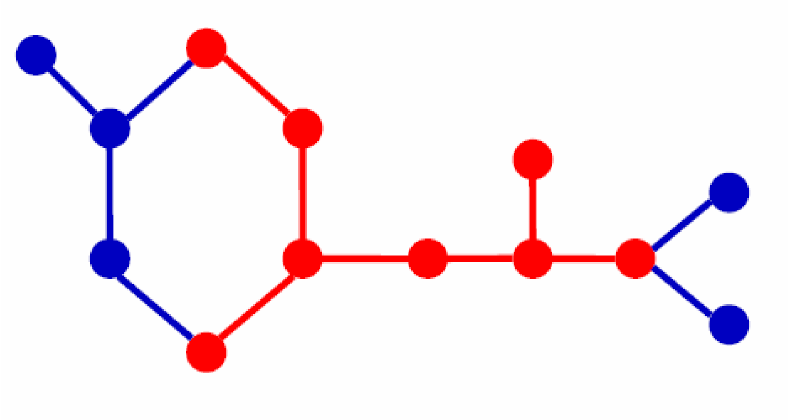
Pattern Growth Method



Graph matching

- Checking if a graph is a subgraph of another graph

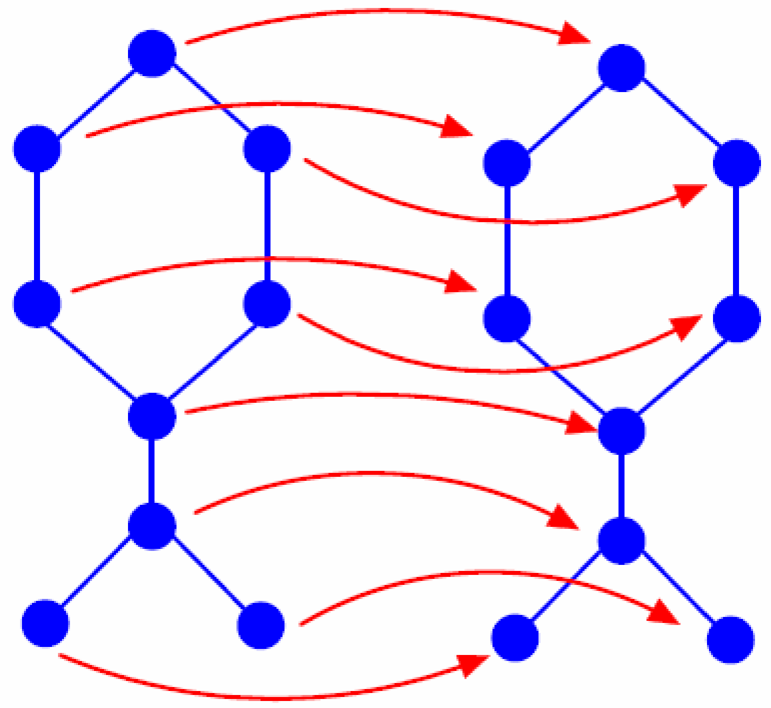




- Why is it crucial?
  - Frequent subgraph patterns require graph matching
  - Labels do not always help
    - Some domains exhibit repeated labels

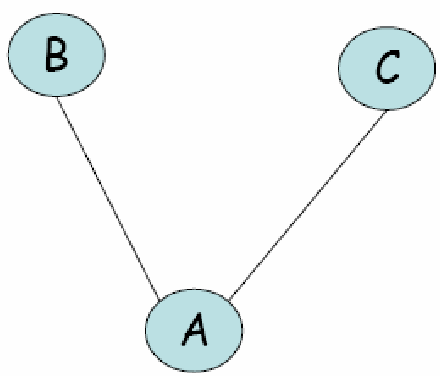
Graph matching (cont.)

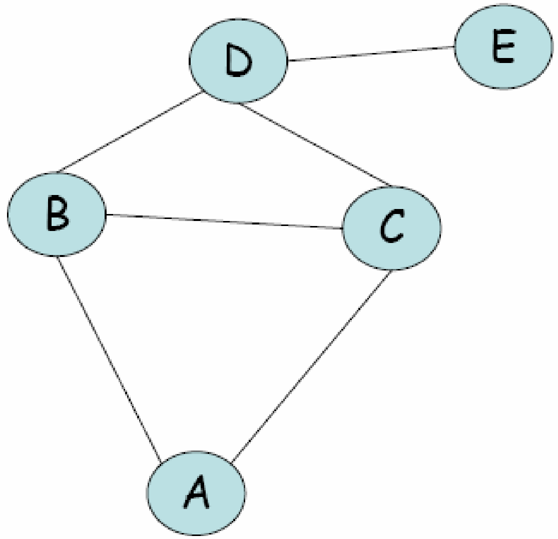
- Must enumerate any possible mapping



(Sub)Graph isomorphism

- There exist a bijective mapping between nodes of the source and nodes of the destination

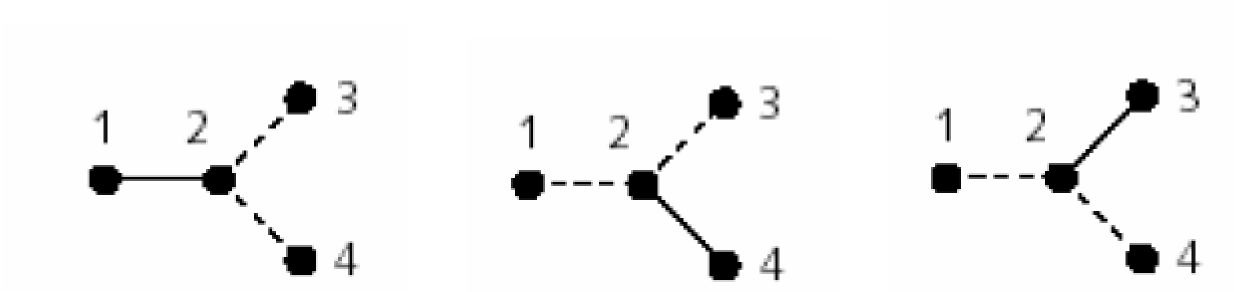




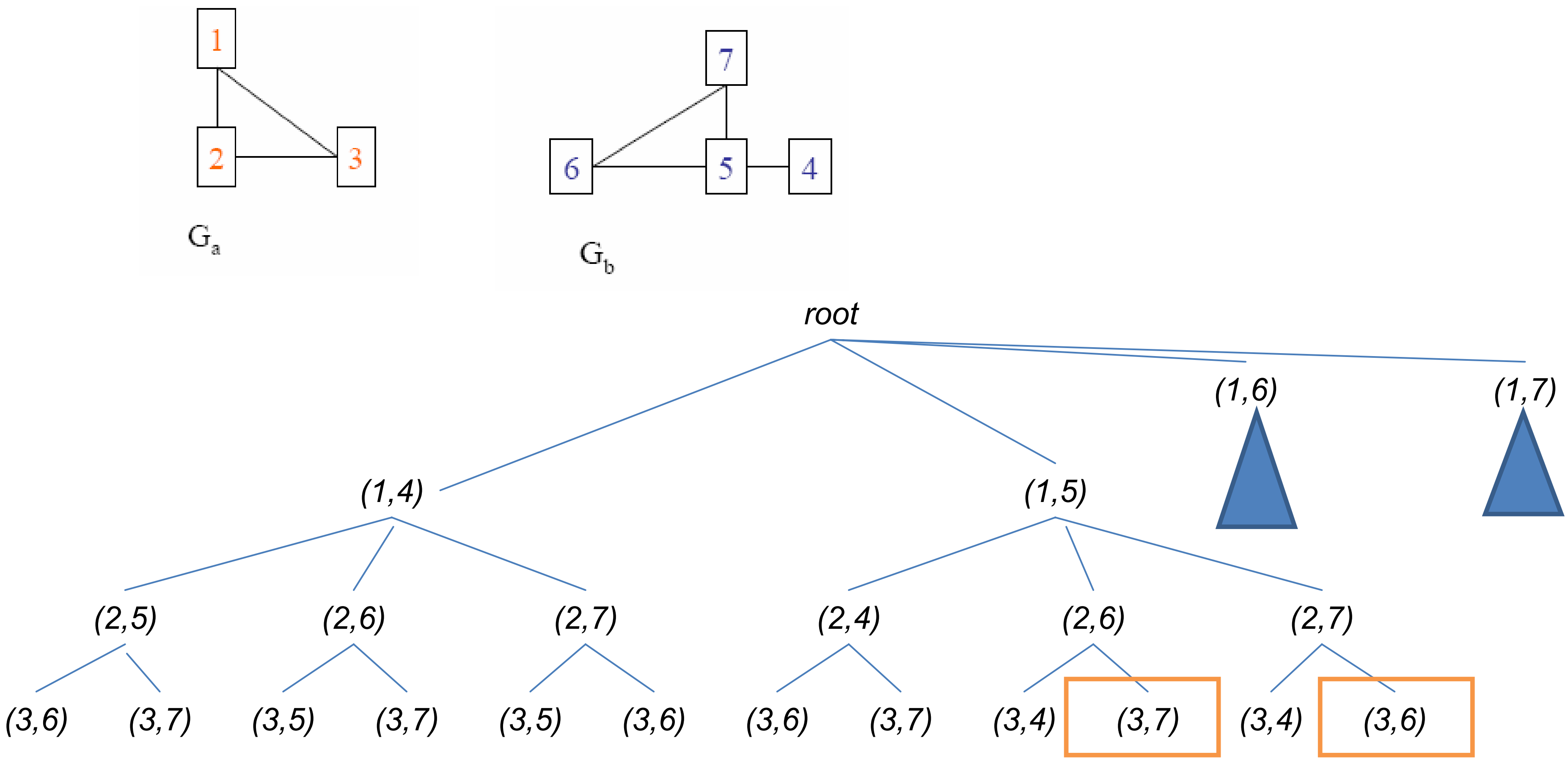
Is that easy?

- Two graphs (always the same label)

- Possible mappings

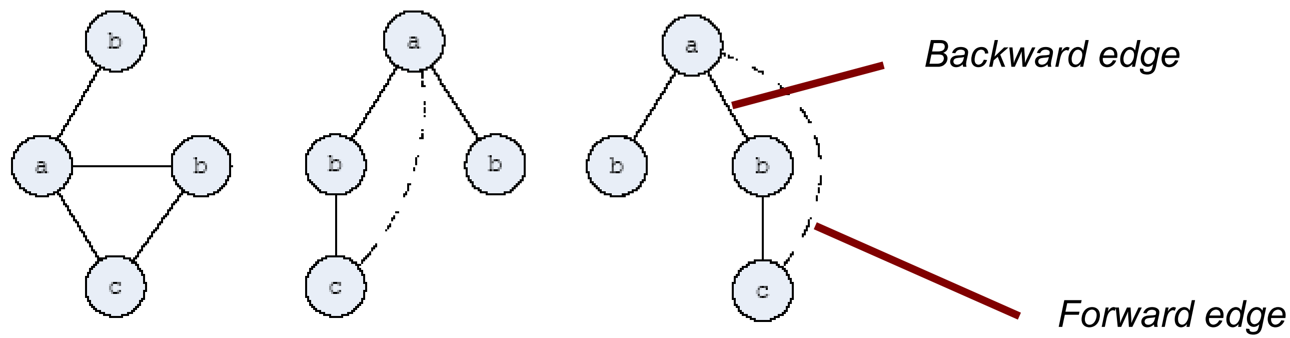


A hard problem



DFS-tree, DFS code

- One graph can have several DFS trees



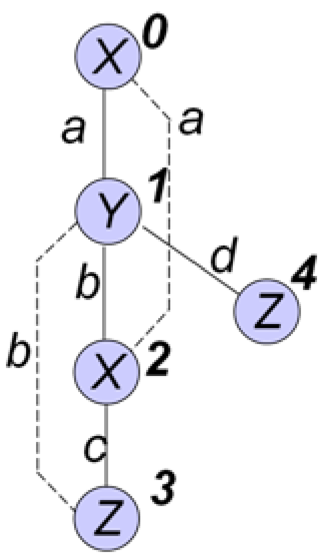
- Each DFS tree can be represented by a sequence of edges-the DFS code

Edge	(b)	(c)
0	(0,1,a,b)	(0,1,a,b)
1	(1,2,b,c)	(0,2,a,b)
2	(2,0,c,a)	(2,3,b,c)
3	(0,3,a,b)	(3,0,c,a)

DFS Code

From DFS codes to canonical forms

- Must choose a canonical form
  - Idea: introduce an order between edges
  - DFS codes are ordered as a consequence
    - The canonical form is the minimum DFS code



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)

Graph Pattern Explosion Problem

- If a graph is frequent, all of its subgraphs are frequent ─ **the Apriori property**
- An **n**-edge frequent graph may have 2^n subgraphs
- Among **422** chemical compounds which are confirmed to be active in an AIDS antiviral screen dataset, there are **1,000,000** frequent graph patterns if the minimum support is 5%

SubGraph Mining Applications

- Classification and Clustering
- Graph Indexing
- Similarity Search

Graph Clustering

- Similarity based approach
  - Feature-based similarity measure
    - Each graph is represented as a feature vector
    - The similarity is defined by the distance of their corresponding vectors
    - Frequent subgraphs can be used as features
  - Structure-based similarity measure
    - Maximal common subgraph
    - Graph edit distance: insertion, deletion, and relabel
    - Graph alignment distance
- Multi-relational approach

Graph Classification

- Graph pattern-based approach
  - Subgraph patterns from domain knowledge
  - Subgraph patterns from data mining
- Kernel-based approach
- Multi-relational approach

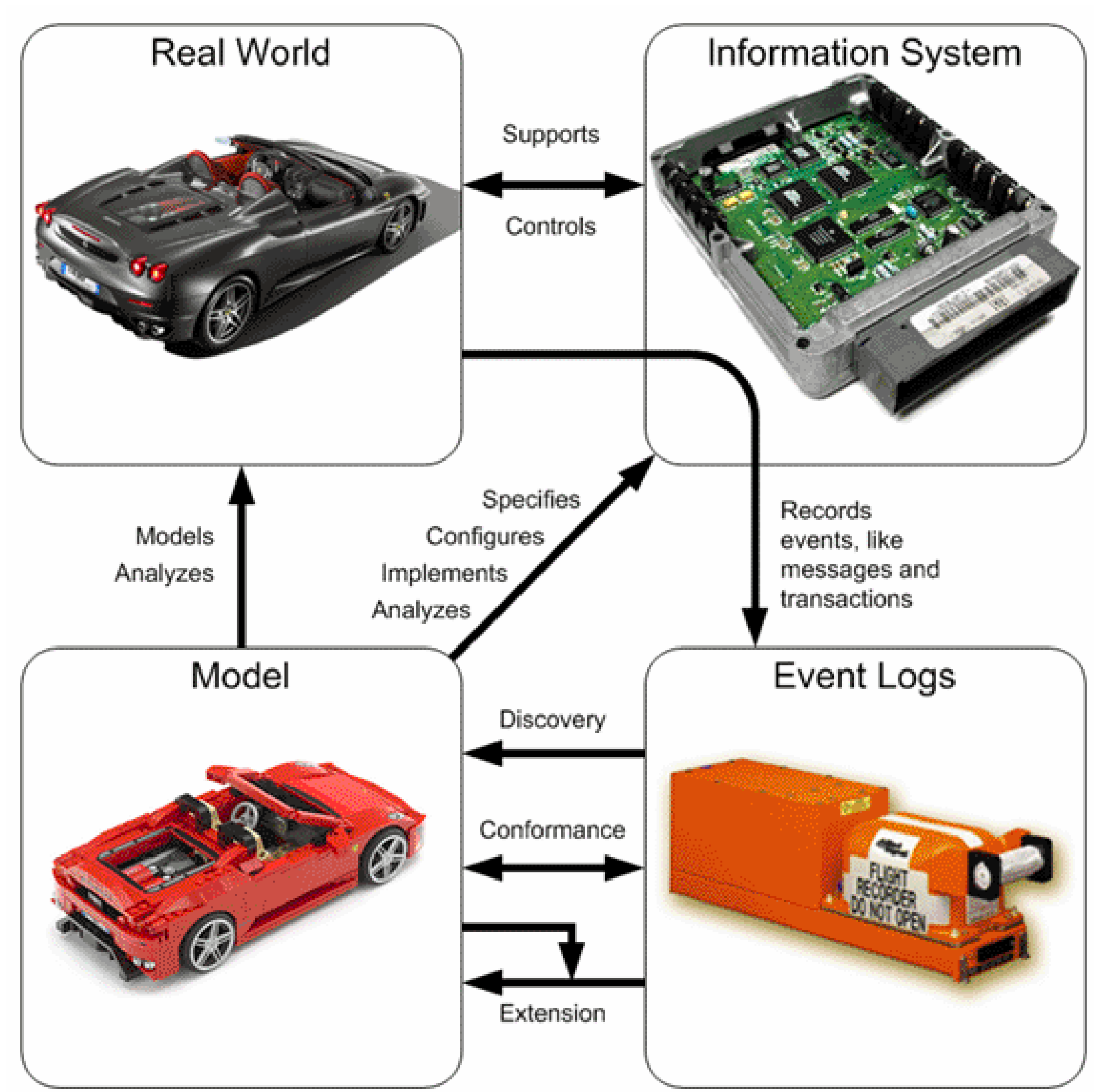
Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Part II:
Workflow Mining

Outline

- Introduzione e concetti di base
  - Motivazioni, applicazioni
  - Concetti di base nell'analisi dei dati complessi
- Text/Web Mining
  - Concetti di base sul Text Mining
  - Tecniche di data mining su dati testuali
- Graph Mining
  - Introduzione alla graph theory
  - Principali tecniche e applicazioni
- Workflow Mining
  - I workflow: grafi con vincoli
  - Frequent pattern discovery su workflow: motivazioni, metodi, applicazioni
- Multi-Relational data mining
  - Motivazioni: da singole tabelle a strutture complesse
  - Alcune delle tecniche principali

Workflows



ICAR CNR

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

A Workflow Process is...

- **The automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for acting, according to a set of procedural rules.** [WFMC]

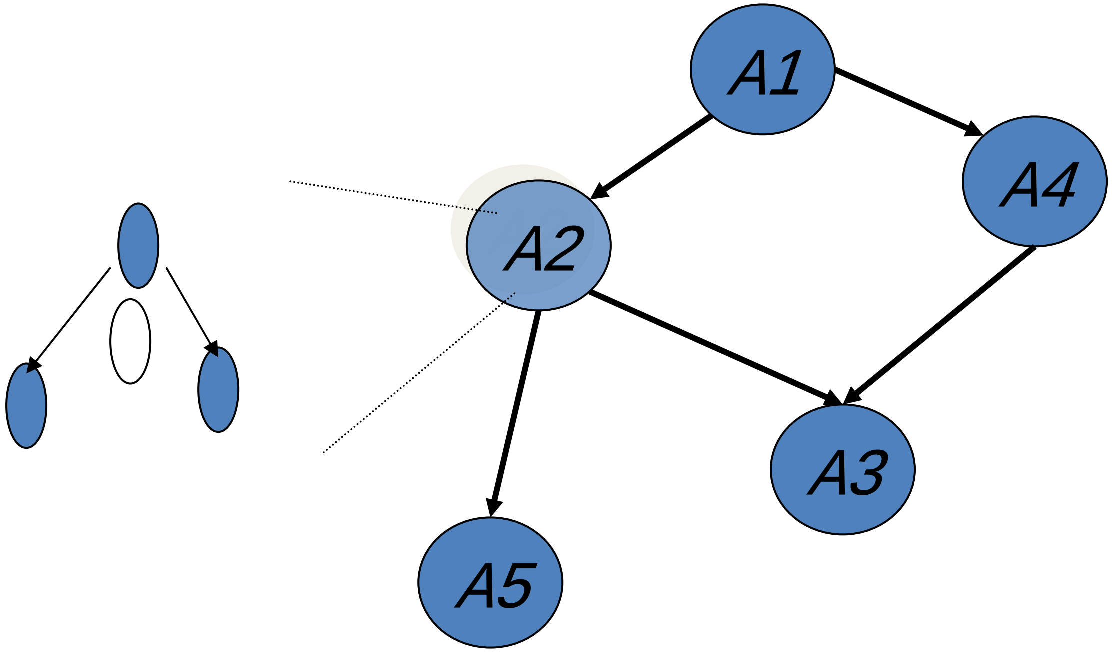
  Example:
  - Credit requests, insurance claims
  - Student exams, journal reviewing
  - Electronic commerce, virtual enterprises, etc.

- We distinguish two aspects:
  - “static”: workflow schema
  - “dynamic”: workflow execution

The Workflow Schema

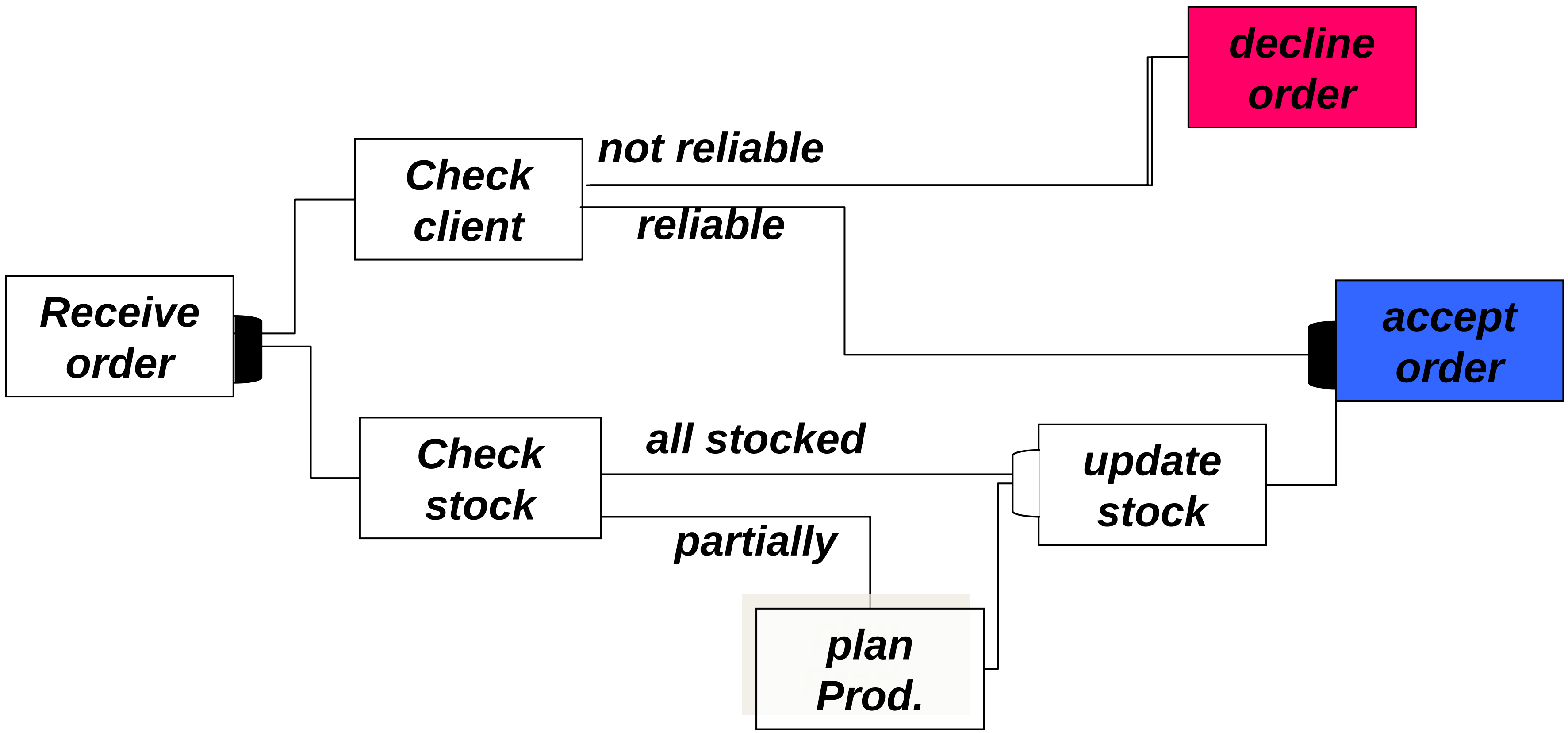
- *Specifies which steps are required and in what order they should be executed*
- *Usually modeled as a directed graph that defines the order of execution among the activities*

Composed by subprocesses and by elementary activities (tasks).

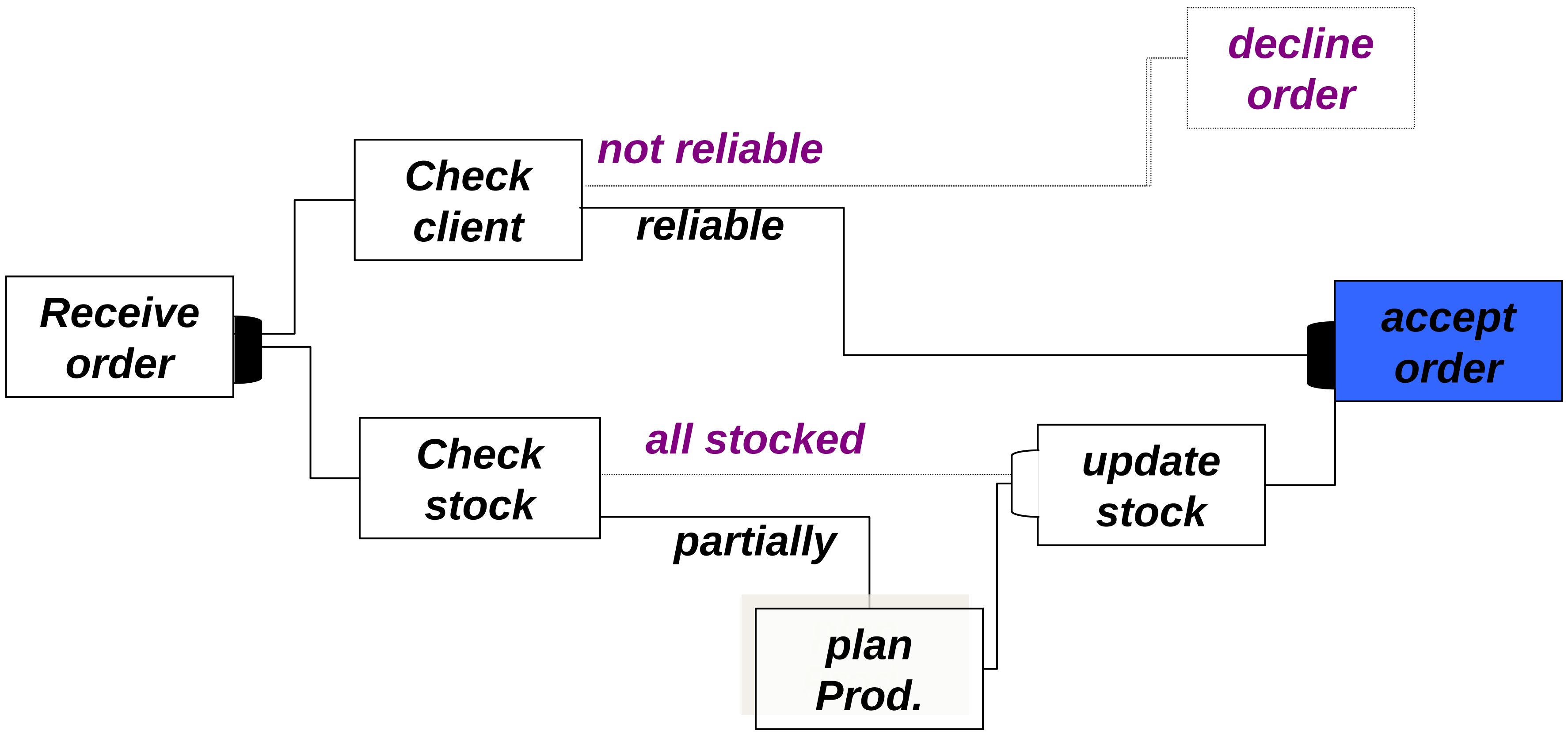


Workflow schema: example

Sales ordering process



Workflow execution: (path)



Source: http://prom.win.tue.nl/research/wiki/processminingcourse

Proce

Event Logs

1. Register at hospital
2. Talk
3. U

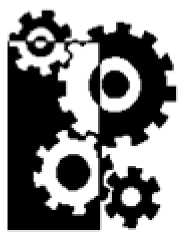
1. Register at hospital

1. Register at hospital
2. Talk to doctor
3. Undergo surgery
4. Take medicines
5. Take blood sample
6. Talk to doctor
7. Go home

Source: http://prom.win.tue.nl/research/wiki/processminingcourse

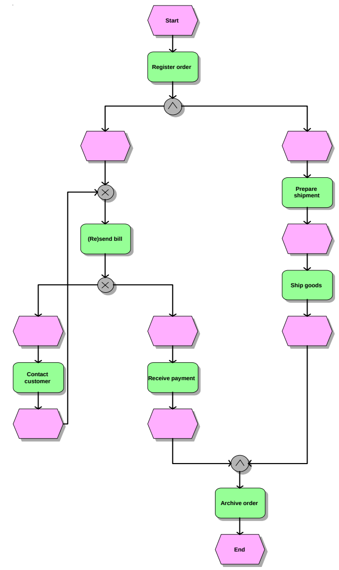
Workflow Mining

Event Log

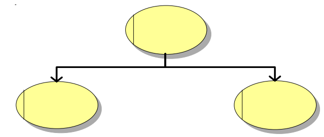


Mining Techniques

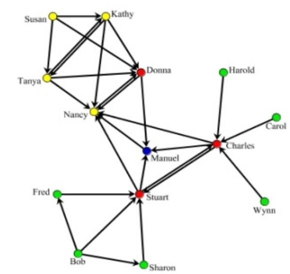
Process Model



Organizational Model

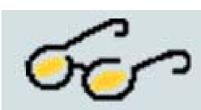


Social Network



Performance Analysis

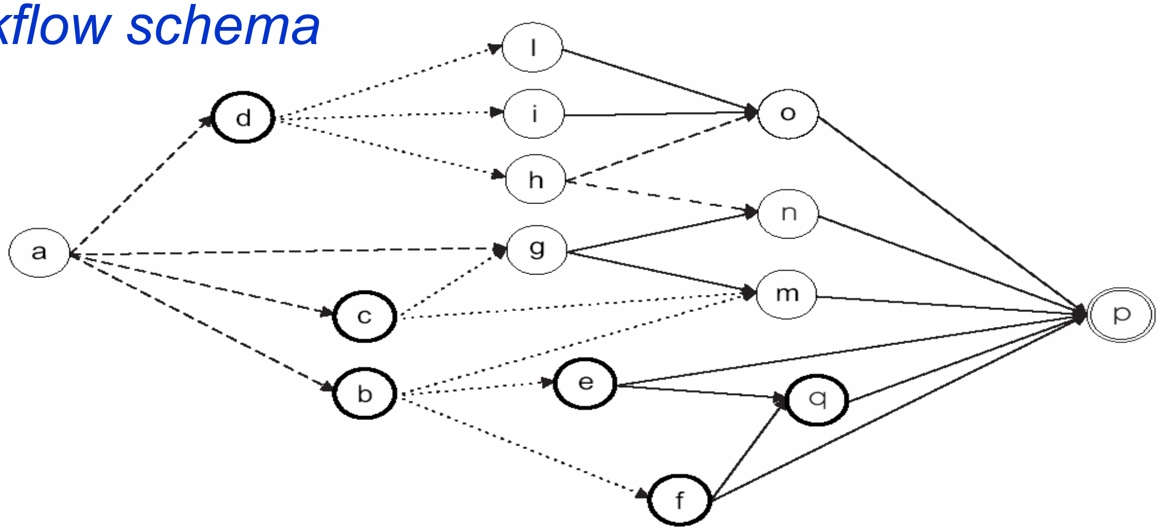
Auditing / Security



Mined Models

Workflow Model

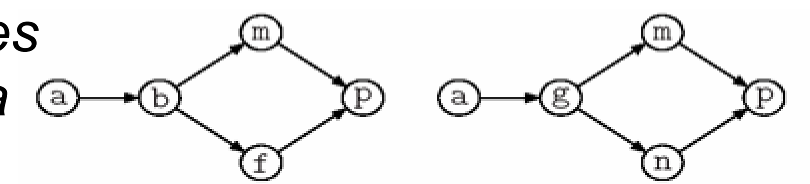
workflow schema

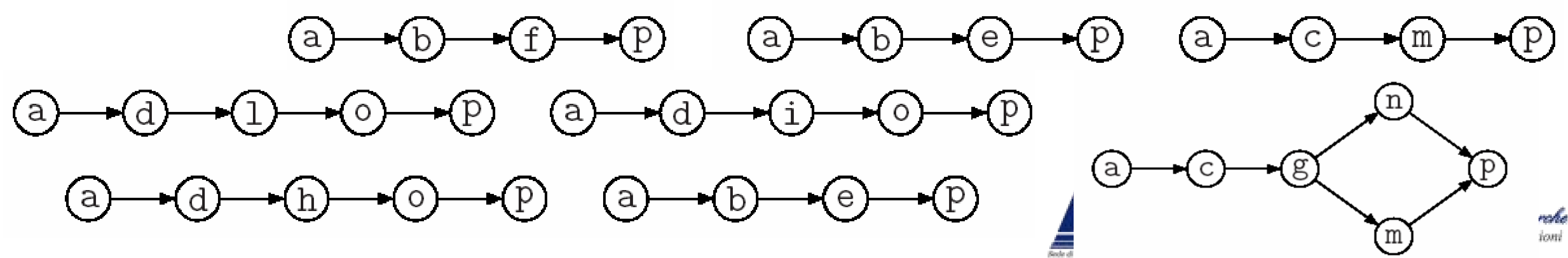


- *directed acyclic graph with constraints modelling an execution*

instances

- *subgraphs of **WS** induced over the nodes corresponding to executed activities in a terminating execution*





Workflow mining

- ***Goal:*** to use the information collected at run-time (collected into log files in any commercial system), in order to:
  - extract unexpected and useful knowledge about the process
  - take the appropriate decisions in the executions of future instances
- ***Main motivation:*** the use of mining techniques is justied by the fact that even “simple” reachability problems become intractable.

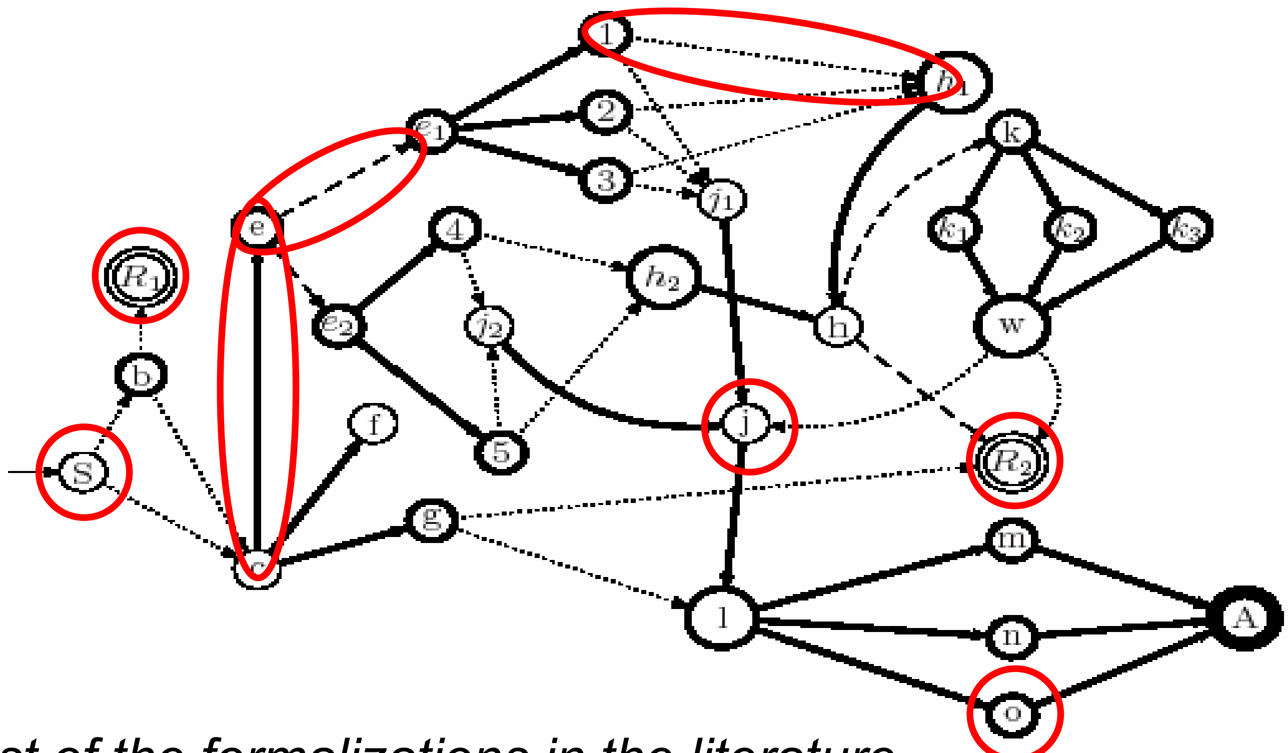
We focus on mining frequent workflow instances (or execution)

Problems that can be solved

- Identification of critical activities
- Failure/success characterization
  - Which discriminant factors characterize the failure or the success in the execution of a workflow
- One Step Prediction
  - Which is the choice performed in the past, that more frequently led to a desired final configuration
- General Graph Prediction
  - Not a singular choice, but a sequence of choices…
- Workflow Optimization
  - We use the frequent structures characterizing the successful execution to reason on the optimality of workflow executions

A workflow abstract model

- *a set of activities, and dependencies among activities*
- *starting activity*
- *final activities*
- *"join" activities*
- *"or" activities*
- *"and" arcs*
- *"xor" arcs*
- *"choice" arcs*



<u>The model covers most of the formalizations in the literature</u>

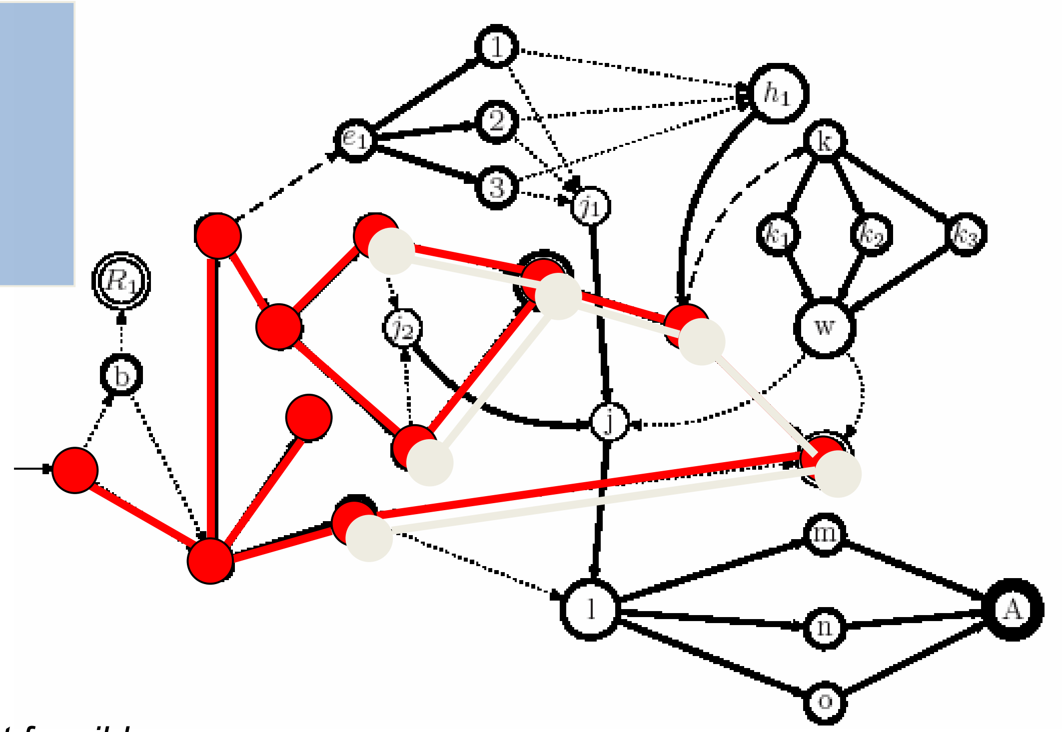
Semantics: complexity

- An instance is a subgraph of the workflow schema induced over the nodes corresponding to executed activities in a terminating execution
- A pattern is a subgraph of an instance

Deciding whether a given subgraph is an instance is in P

… but

Deciding whether a given subgraph is a pattern is NP-complete



<u>"Static" reasoning is not feasible</u>

Mining frequent patterns

- Let F={I_1,…,I_n} be a set of instances. Then, a F-pattern p is a subgraph of some instance in F
- The support of an F-pattern is the fraction of the instances in F in which is contained.
- Let minSupp be a real number.

Find the set of all the F-pattern whose support is greater then minSupp

A naive solution is to model the problem in terms of relational database and use the Apriori algorithm

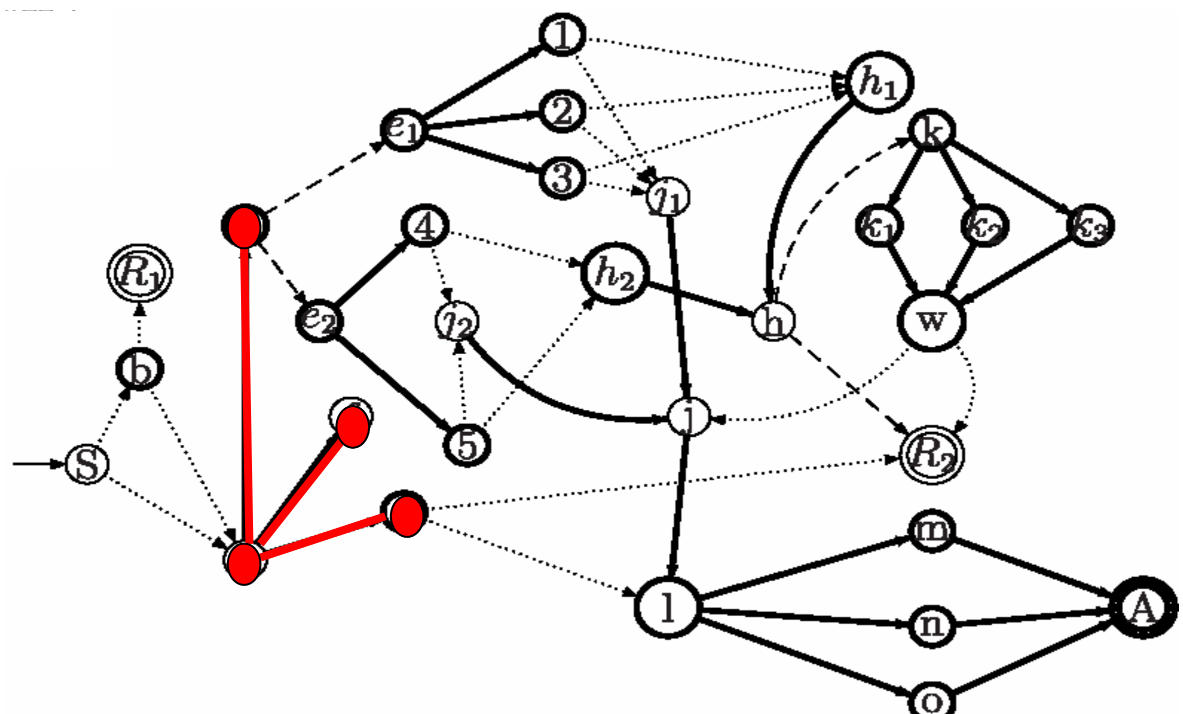
More sophisticated algorithms can exploit the graphical structures and the peculiarity of the workflows

(1a) Mining connected patterns

- Frequent connected F-pattern:
  - the undirected subgraph must be connected
- Deterministically closed:
  - all the incoming arcs in an “join” node of a pattern are in the pattern, too
  - all the “and” arcs outcoming from a node of a pattern are in the same pattern, too.

Mining connected patterns

- Working “directly” with patterns is unfeasible
- Deterministic closed patterns
    - all the incoming arcs in an “join” node of a pattern are in the pattern, too
    - all the “and” arcs outcoming from a node of a pattern are in the same pattern, too.



Weak patterns

- A weak pattern is a connected and deterministic closed pattern
  - Deciding whether a graph is a weak pattern is "difficult"
  - Each frequent F-pattern can be obtained by composing frequent weak patterns

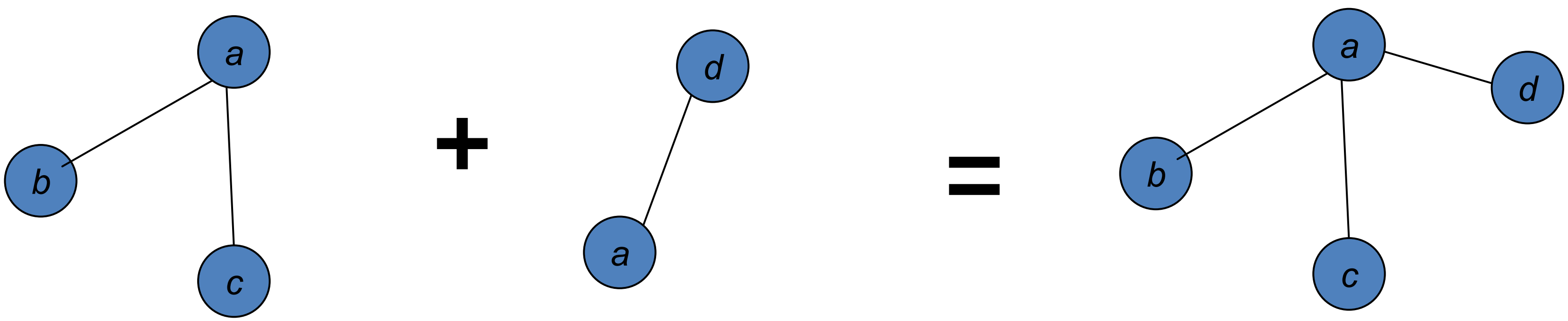
The set of all the weak patterns is a lower semi lattice w.r.t. to a suitable precedence operator, and, hence, can be explored in a level-wise fashion.

Elementary weak patterns are the closures of single nodes.

ICAR CNR
Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

The W-find algorithm

- start from frequent "elementary" weak patterns
- extend elementary patterns using two basic operations:
    - adding a frequent arc
    - merging with another frequent elementary weak pattern.



L_k Ew L_{k+1}

The W-Find algorithm (II)

Let L_0 be the set of "elementary" weak patterns
repeat
  U:=0
  forall p in L_k do
    U:=U + addFrequentArc(p)
    forall e in L_0 do
        U:=U + addFrequentEWPattern(p,e)
  endfor
 L_{k+1} :=L_k + frequentPatternIn(U)   //scan into the DB
 R:=R + L_{k+1}
until L_{k+1}=0
return R

Sound and complete w.r.t. the set of all F-patterns

(1b) Mining disconnected patterns

Finding frequent maximal disconnected F-patterns

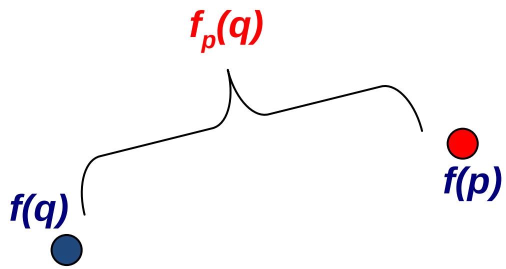
- Basic Algorithm
  - Starting from frequent connected F-patterns containing the start activity
  - Constructing a frequent disconnected F-pattern P by adding frequent connected F-patterns which are not connected to P

Mining disconnected patterns
An improved Algorithm

- Constructing the frequency graph: counting the nodes and the edges in all instances
- Computing frequency bounds for activities
  - given a node *a* with frequency f(*a*)
    - for each node *b* preceding *a*
  - find lower bound and upper bound on the number of instances containing both *a* and *b*

Upper and *Lower* bounds

- ***q*** *is a not-necessary connected component with frequency* ***f(q)***
- ***p*** *is a connected component with frequency* ***f(p)***
- $f_p(q)$ *is the number of istances in F executing both the component p and q*



Idea:

exploit both the workflow structure and frequency of each activity to identify, before their actual testing w.r.t. workflow logs, those patterns which are necessarily (un)frequent.

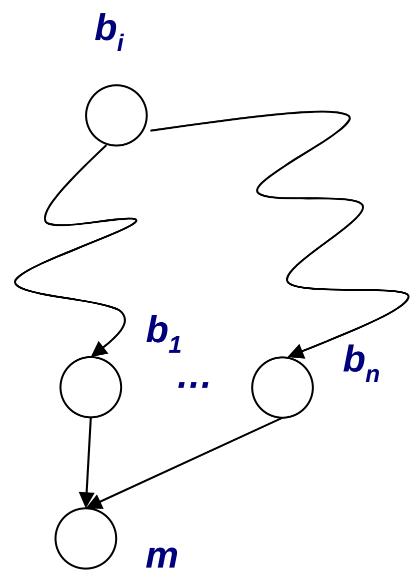
Upper and Lower *bounds on* $f_p(q)$ *are used for efficiently pruning the search space*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Computing Frequency bounds for activities

For each activity **m** in **WS**

1. starting from a topological sort $<m, b_1, b_2, \ldots b_n >$
2. we compute for each node b_i, $l_m(b_i)$ and $u_m(b_i)$



Computing Frequency bounds for activities

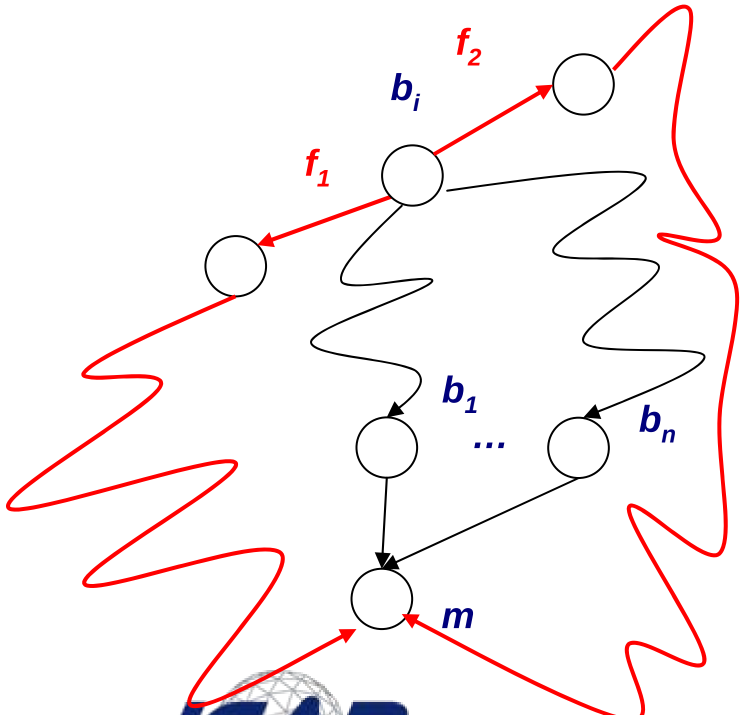
For each activity **m** in **WS**

1. starting from a topological sort $<m, b_1, b_2, \ldots b_n>$
2. we compute for each node b_i, $l_m(b_i)$ and $u_m(b_i)$

Computing $u_m(b_i)$

- optimistic assumption that each arc outgoing from b_i is in some path reaching m
- contributes to frequency of $\mathbf{b_i}$ by adding its frequency $\mathbf{f_e}$

$$u_m(b_i) = f_1+f_2$$



Computing Frequency bounds for activities

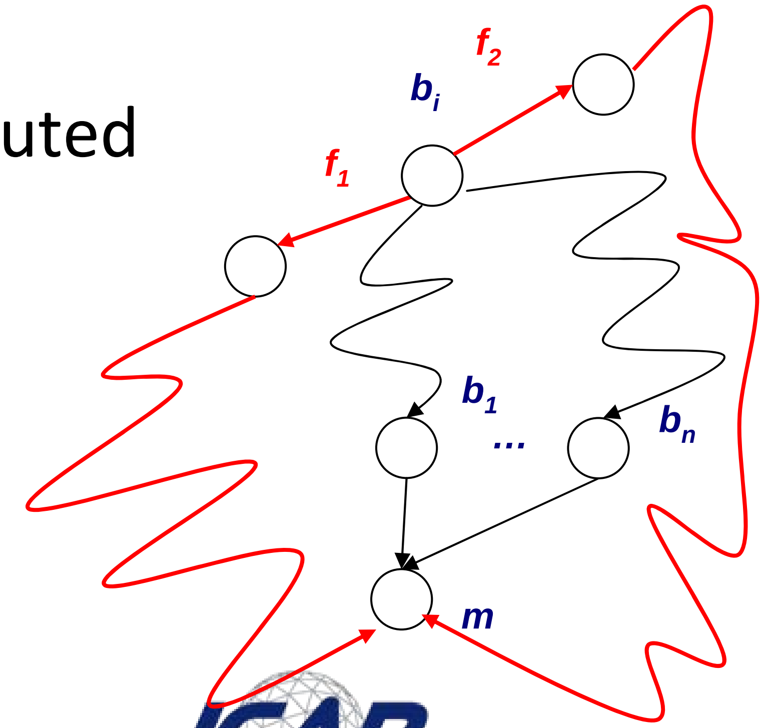
For each activity **m** in **WS**

1. starting from a topological sort $<m, b_1, b_2, \ldots b_n >$
2. we compute for each node b_i, $l_m(b_i)$ and $u_m(b_i)$

Computing $l_m(b_i)$

- XOR-split node
  - Nodes connected to $\mathbf{b_i}$ are executed exclusively
  - The single contributions are summated

$$l_m(b_i)= f_1+f_2$$



Computing Frequency bounds for activities

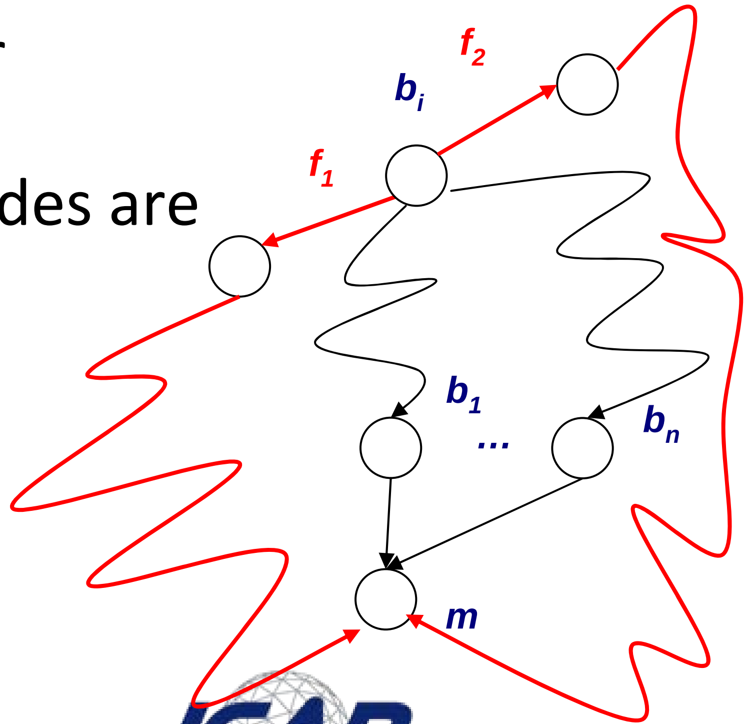
For each activity **m** in **WS**

1. starting from a topological sort <m, b_1, b_2, ...b_n >
2. we compute for each node b_i, $l_m(b_i)$ and $u_m(b_i)$

Computing $l_m(b_i)$

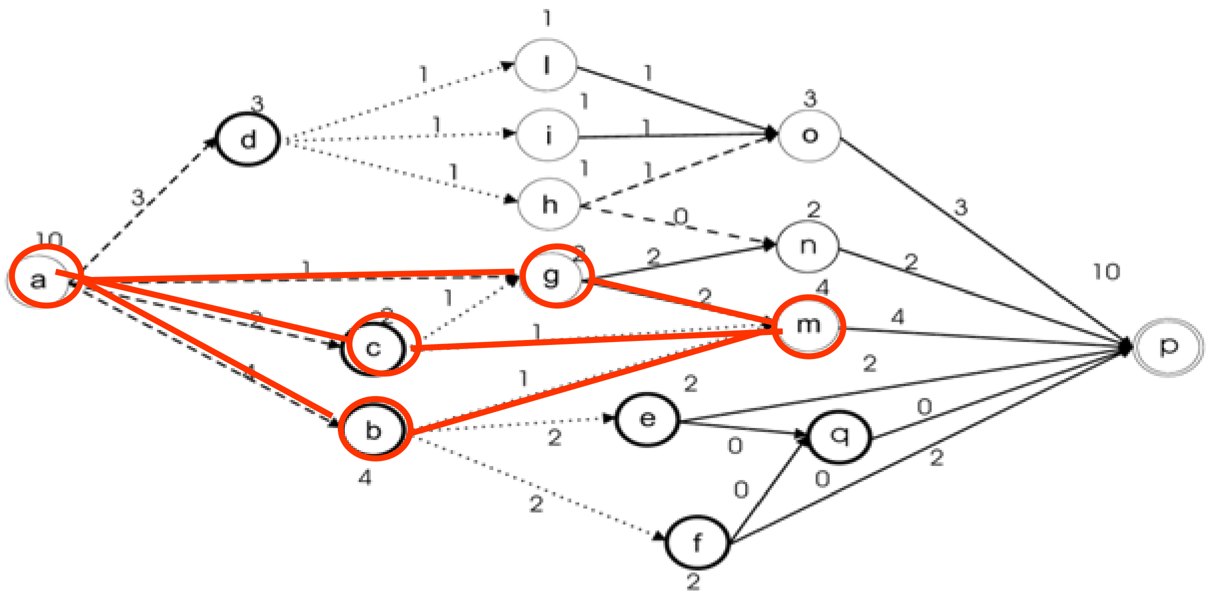
- OR-split/AND-split node
  - nodes connected to $\mathbf{b_i}$ may occur simultaneously
  - The contributions of adjacent nodes are maximized

$$l_m(b_i) = max(f_1, f_2)$$



Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Computing Frequency bounds for activities



$l_m(g) = 2$, $u_m(g) = 2$, $l_m(b) = 1$, $u_m(b) = 1$,
$l_m(c) = 1$, $u_m(c) = 2$, $l_m(a) = 3$, $u_m(a) = 4$