



JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

Data Mining – Graph Mining

Andreas Karwath Jörg Wicker

Johannes Gutenberg University Mainz

March 21, 2017

Outline

- Graph Mining - Mining Graph Databases
- GSPAN

Section 1

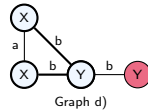
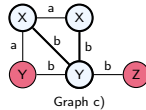
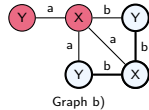
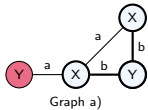
Graph Mining - Mining Graph Databases

Motivation

- Applications in bioinformatics/cheminformatics:
 - finding network motifs in metabolic and regulatory networks
 - finding interesting substructural fragments in databases of small molecules
- Interesting (frequent) subgraphs can then be turned into features for classification tasks

Graph and Subgraph Isomorphism - Motivation

- Given the following graph database, how often does the pattern p_1 match the examples?



- Pattern p_1 :

- Commonly known under the name : graph or sub-graph isomorphism.

Graph and Subgraph Isomorphism

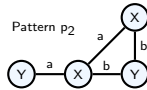
- Definition: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, denoted $G_1 \cong G_2$, if there is a bijection $\varphi : V_1 \rightarrow V_2$, such that, for every pair of vertices $v_i, v_j \in V_1$, $(v_i, v_j) \in E_1$ if and only if $(\varphi(v_i), \varphi(v_j)) \in E_2$
- Definition: A graph $G_1 = (V_1, E_1)$ is isomorphic to a subgraph $G_2 = (V_2, E_2)$, denoted $G_1 \cong S_2 \subseteq G_2$, if there is an injection $\varphi : V_1 \rightarrow V_2$, if $v_i, v_j \in V_1$, $(v_i, v_j) \in E_1$ then $(\varphi(v_i), \varphi(v_j)) \in E_2$

Graph and Subgraph Isomorphism

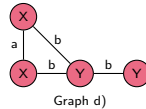
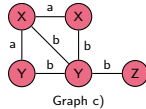
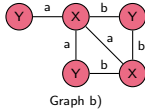
- Graph isomorphism: complexity for best possible algorithm not yet known
- Fastest isomorphism testing program is Nauty, due to Brendan D. McKay
- Subgraph isomorphism: classic NP-complete problem
- For instance, algorithm by Ullmann, not brute- force tree-search enumeration procedure, but eliminating successor nodes in the tree search.

Graph and Subgraph Isomorphism

- Subgraph pattern p_2 :

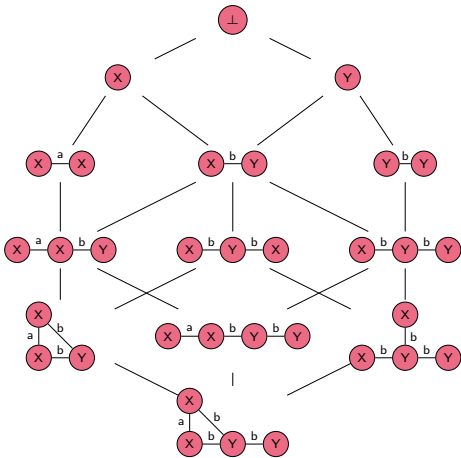
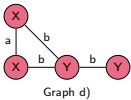


- Database D consisting of graphs $b)$, $c)$, and $d)$:

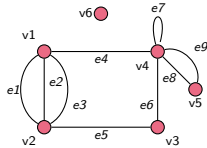


- $\text{support}(p_2, D) = 2$

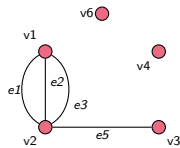
Lattice of All Subgraphs of Graph d)



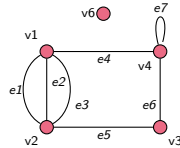
Types of Subgraphs



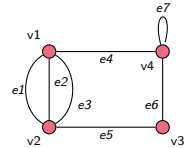
(a) A graph



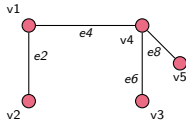
(b) A general subgraph



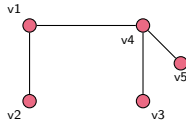
(c) An induced subgraph



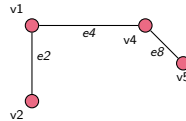
(d) A connected subgraph



(e) An ordered tree

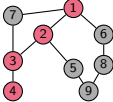
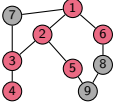
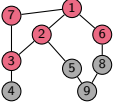


(f) An unordered tree



(g) A path

Graph Mining

	Paths	Free Trees	Subgraphs
Example			
Subsumption	linear	$O(\frac{k^{1.5}}{\log k} n)$	exponential (NP-complete)
Coverage	exponential (NP-complete)	exponential (NP-complete)	exponential (NP-complete)
Canonization	linear	linear	graph isomorphism

Section 2

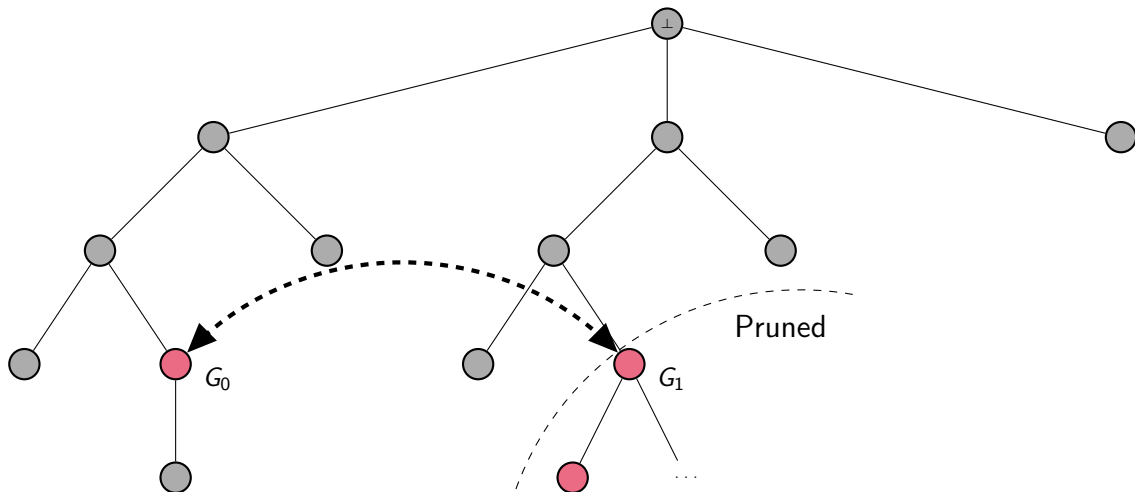
Graph Mining - GSPAN

GSPAN

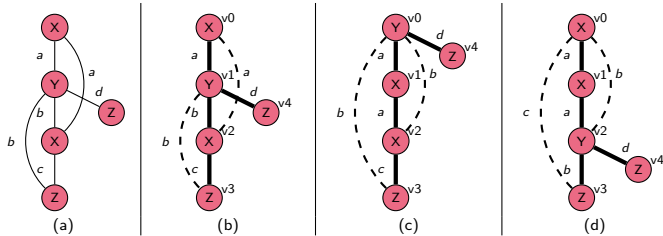
- Due to (Yan & Han, 2002)
- Canonical form: depth-first encoding of spanning trees of the graphs
- Depth-first traversal of search tree, *enumerate all codes in lexicographic order*
- Search only in the space of canonical forms (list of computationally cheap *necessary* conditions for a graph encoding to be in canonical form)
- The first time a graph (node in the search tree) is encountered, its encoding is *automatically* in canonical form - for any further encounters, it *cannot* be in canonical form which can be checked efficiently in most cases

$$\text{Canonical} \subseteq \text{Generated}$$

Search Tree and Canonical Forms



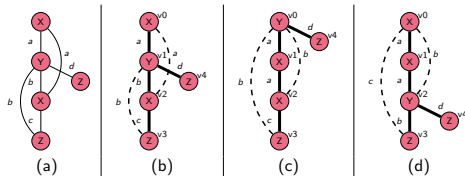
Canonical Form for Spanning Trees



- So-called DFS code with forward (—) and backward (---) edges
- (c) has second node from top as root
- (d) has third node from top as root

Minimal DFS Code

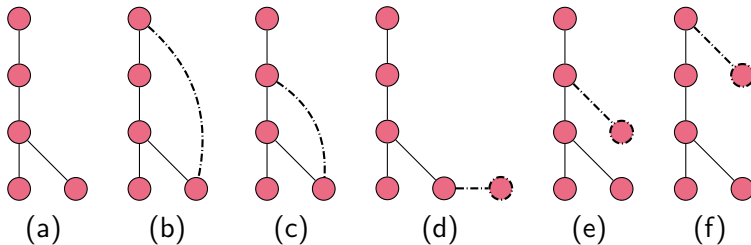
The minimum DFS code $\min(G)$, in some defined order, is the canonical label of graph G .



edge	α (Fig 1b)	β (Fig 1c)	γ (Fig 1d)
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)

Order on DFS Codes

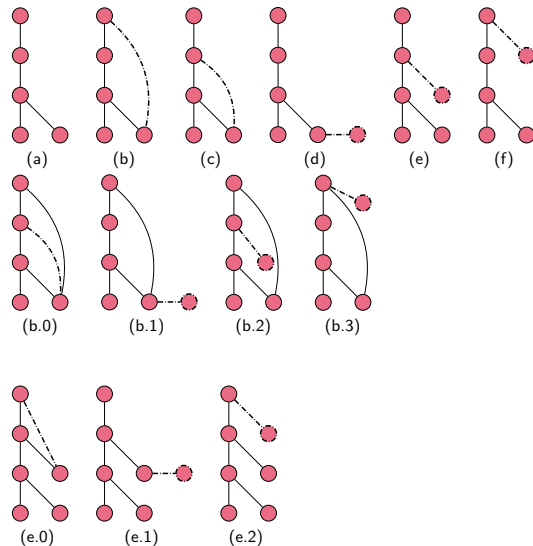
- According to insertion position of edges
- Edge type (backward/forward edge)
- Node and edge labels



Canonization

- Need not be done explicitly
- If an arbitrary graph had to be encoded, we would have to choose any node as the root, then generate all possible spanning trees, then take the smallest of them lexicographically
- However, the graph encodings are already created in canonical form (or immediately discarded if check reveals it is not canonical) that is, we do not have to encode the graphs at all - we are only working with the codes themselves

Search Tree gSpan



- Only expand on the right-most path
- Backward edges from right-most node
- Forward edges from right-most path
- All possible extensions by forward or backward edges suggested by the embeddings in the graph data are generated

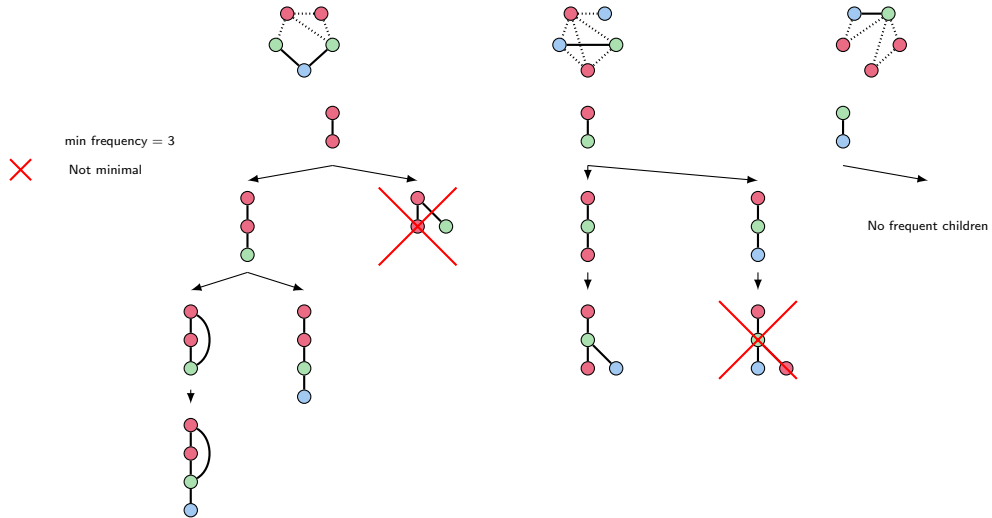
Search Tree gSpan

- Remove infrequent vertices and edges from database
- Relabel in *descending* frequency
- Find all frequent one-edge subgraphs S_1
- Iterate over all elements s from S_1 :
 - test if DFS code of s minimal
 - if yes, find all extensions of s in a database scan
 - iterate over all frequent extensions to determine their extensions
 - ... recursively extend until no extension can be found (depth-first search)
 - remove edges of type s from database

Database Scan: Subgraph Isomorphisms and Extensions

- p to be tested against database $D = g_1, g_2, g_3, g_4$
- Existing extensions of $p : p_1, p_2, p_3, p_4, p_5$
- After g_1 : $[p_1 : 1, p_2 : 1, p_3 : 1]$
- After g_2 : $[p_1 : 2, p_2 : 1, p_3 : 1, p_4 : 1]$
- After g_3 : $[p_1 : 3, p_2 : 1, p_3 : 1, p_4 : 2, p_5 : 1]$
- After g_4 : $[p_1 : 3, p_2 : 2, p_3 : 1, p_4 : 3, p_5 : 2]$
- After g_4 : $[\underline{p_1 : 3}, p_2 : 2, p_3 : 1, \underline{p_4 : 3}, p_5 : 2]$

Example



Important Detail

- The *pattern growth* idea applied here implies that **all** subgraph isomorphisms have to be computed in gSpan
- In AGM, only the first needs to be computed, since we are just counting the number of occurrences

gSpan

- Space-efficient by virtue of depth-first search
- Candidate generation and testing *integrated*: keeping and extending embeddings of graphs (extension only on the *right-most path*, forward)
- Data simplified during search: after branch has been searched completely, the corresponding parts of the graph data can be removed
- No frequency-based pruning (guarantee of at least one frequent predecessor graph only)

Conclusion Graph Mining

- Based on ideas known from item set, sequence, and tree mining
- Canonical forms are even more essential
- Again, breadth-first vs. depth-first: a trade-off
- Keeping embeddings or not?
- Comparison (Worlein *et al.*, 2005) showed: gSpan still competitive
- Generalization to hypergraphs, multi- label graphs, numerical labels, ...