Graph pattern mining

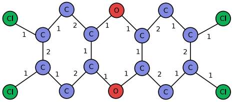
Francesco Bariatti

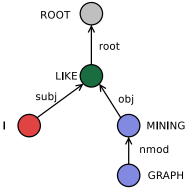
francesco.bariatti@irisa.fr

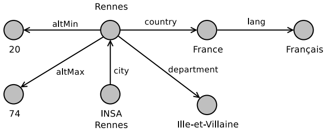
04/01/2021

- Graph $G = (V, E)$: data structure with a set of *vertices* V and a set of *edges* $E \subseteq V \times V$ connecting them
  - *Undirected* graph: edges (u, v) and (v, u) are the same.
  - *Labeled* graph $G = (V, E, l)$: labeling function l associating labels to vertices and edges.
  - Most graph mining approaches focus on undirected labeled graphs.
- Graphs are sometimes called *networks* depending on the domain

- Graphs are a powerful and expressive structure to represent data, used in many domains: molecules, physical networks (telco), social networks, text corpora, program traces (call graph), semantic web...

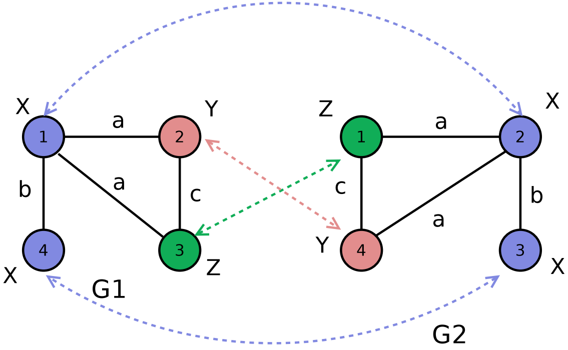






(Sub)graph isomorphism

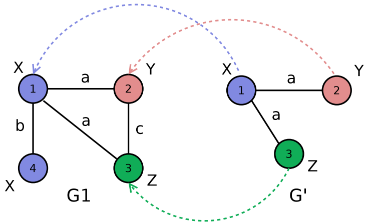
- Graphs can appear different but actually have the same structure
- *Graph isomorphism*: recognizing if two graphs are the same
  - Isomorphism: bijective function mapping vertices of G1 to vertices of G2 so that edges and labels are preserved



G1 and G2 are isomorphic

(Sub)graph isomorphism

- *Subgraph isomorphism*: recognizing if a graph is part of another graph
  - A graph G' is subgraph isomorphic to a graph G if there exist an injective function $\varepsilon \in V' \to V$ such that $\forall e = (u, v) \in E' \ \ (\varepsilon(u), \varepsilon(v)) \in E$; $\forall v \in V' \ \ l(\varepsilon(v)) = l(v)$; $\forall e \in E' \ \ l(\varepsilon(e)) = l(e)$.
  - We call ε an *embedding* or *occurrence* of G' in G



G' is a subgraph of G1

- Subgraph isomorphism search is NP-complete!
  - In practice if labels are diverse enough, it can be computed in reasonable time. *But sometimes data does not play nice*

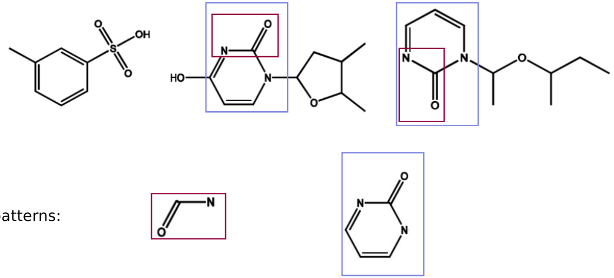
Graph pattern mining

Graph pattern mining is essentially the problem of discovering frequent subgraphs (patterns) occurring in the input data graph(s).

- Find structures describing interesting concepts in the data
- Abstract parts of the data as instances of patterns
- Learn about the data by looking at what is frequent in it

Data:

Example patterns:



What is frequent?

- Discovering frequent subgraphs = discovering subgraphs with a support greater than user-given parameter *minsup*
- Support definition depends on the kind of graph data
  - Base idea similar to other pattern mining domains: "how often is the pattern found in the input data?"

Two families of graph data:

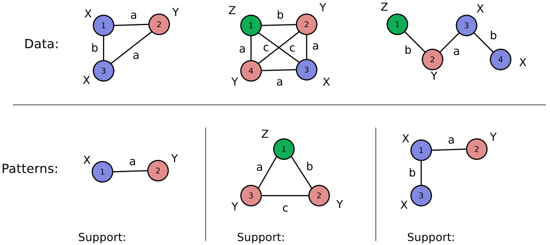
- Graph collection: a (generally large) set of (generally small) graphs.
  - e.g. molecules, sentences
- Single graph: the data is a unique (generally large) graph
  - e.g. semantic web, social networks, DNA

What is frequent in a graph collection?

Let $\mathcal{D}$ be a graph collection and P a graph pattern,

$$support(P) = \frac{|\{g \in \mathcal{D} \mid P \text{ is subgraph-isomorphic to } g\}|}{|\mathcal{D}|}$$

- Each graph of the collection can only contribute once to the support, even if it has multiple occurrences of the pattern

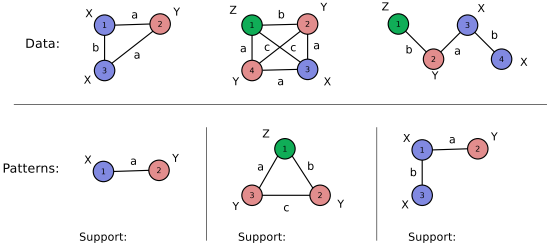


What is frequent in a graph collection?

Let $\mathcal{D}$ be a graph collection and P a graph pattern,

$$support(P) = \frac{|\{g \in \mathcal{D} \mid P \text{ is subgraph-isomorphic to } g\}|}{|\mathcal{D}|}$$

- Each graph of the collection can only contribute once to the support, even if it has multiple occurrences of the pattern

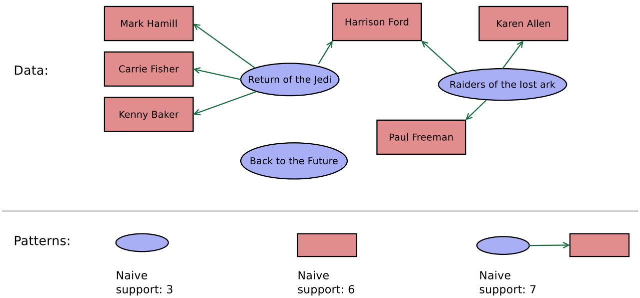


- This measure is **anti-monotonic**: support of a graph is lower or equal to support of its subgraphs

What is frequent in a single graph?

Naive solution

Count how many occurrences the pattern has in the graph.



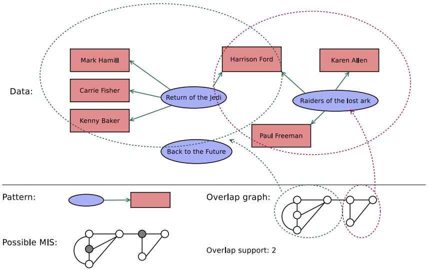
Naive support is **not** anti-monotonic

What is frequent in a single graph?

Overlap-based approaches [Kuramochi and Karypis, 2004]

1. Compute overlap graph of pattern embeddings
2. Support is size of MIS (Maximum Independent Set) of overlap graph

- i.e. maximum number of non-overlapping embeddings of the pattern



Overlap-based support **is** anti-monotonic

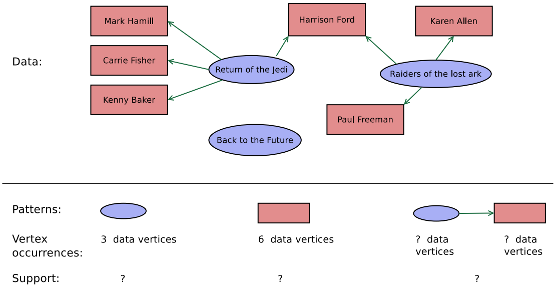
MIS computation is NP-complete

What is frequent in a single graph?

Minimum image based support [Bringmann and Nijssen, 2008]

Let $\mathcal{D}$ be a single data graph and P a graph pattern,

$$support(P) = \min_{v \in V^P} |\{\varepsilon(v) \mid \varepsilon \text{ is an embedding of } P \text{ in } \mathcal{D}\}|$$



Minimum image based support **is** anti-monotonic
Does not need to compute a NP-complete problem

Pattern-merging algorithms

If the support measure is anti-monotonic, for a k-size pattern to be frequent, all its (k-1)-size elements must be frequent.

Pattern-merging graph mining algorithms work similarly to Apriori:

0. Given L_k the set of k-size *frequent* patterns
1. Merge *compatible* k-size patterns to create C_{k+1} the set of candidate (k+1)-size patterns
   - Compatible k-size patterns: patterns that have a common (k-1)-size core (i.e. differ in only one element)
2. Prune C_{k+1}: only retain patterns whose *all* (k-1)-size elements are frequent
3. Create L_{k+1} by computing support of all patterns in C_{k+1}
   - If $L_{k+1} = \emptyset$, stop

- Main pattern-merging graph mining algorithms:
  - AGM/AcGM [Inokuchi et al., 2000]
  - FSG [Kuramochi and Karypis, 2001]
  - DPMine [Gudes et al., 2006]
- Main difference is the definition of a k-size pattern: k vertices, k edges, k edge-disjoint paths, . . .

Drawbacks of pattern-merging approaches

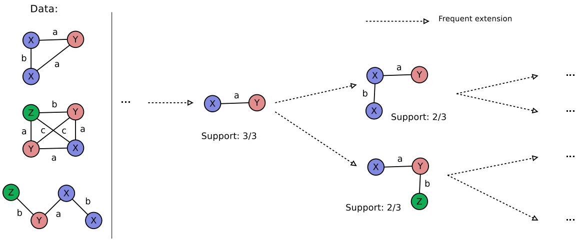
- Merging k-size patterns to create (k+1)-size pattern requires finding patterns that share a common (k-1)-size core $\rightarrow$ subgraph isomorphism
- Pruning step: need to verify if (k-1)-size elements of a pattern are frequent $\rightarrow$ subgraph isomorphism
- Support computation: need to find occurrences of pattern $\rightarrow$ subgraph isomorphism
  - Can be skipped by storing information in memory $\rightarrow$ memory consumption
- Breadth-First Search (BFS): need to store all frequent k-size patterns $\rightarrow$ high memory consumption

Remember that subgraph isomorphism is NP-complete!

Pattern-growth algorithms

Solve drawbacks of pattern-merging algorithms:

- Expand frequent patterns by looking at possible frequent extensions of their embeddings
    - No need to merge patterns → avoid subgraph-isomorphism check: time gain
    - No need to store all k-size patterns to generate (k+1)-size patterns: memory gain
    - Only generate frequent patterns → avoid testing non-frequent candidates: time gain
- Most algorithms in this family use depth-first search to generate patterns → often called DFS algorithms



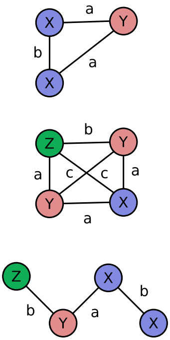
Pattern-growth algorithms

Most used/cited pattern-growth algorithms [Wörlein et al., 2005]:

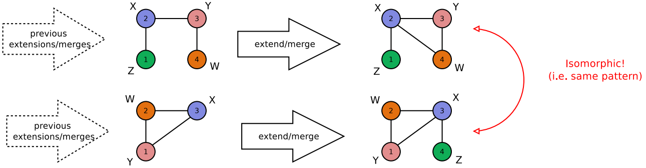
- MoFa [Borgelt and Berthold, 2002]
  - Developed to find substructures in collection of molecules
  - Least efficient of the four because it generates many times the same patterns
- gSpan [Yan and Han, 2002]
  - The most cited
  - Introduces techniques to avoid generating multiple times the same patterns (canonical labeling, DFS with rightmost path expansion)
- FFSM [Wang et al., 2003]
  - Uses both pattern extension and a special efficient join operation
- Gaston [Nijssen and Kok, 2005]
  - Works in phases to avoid subgraph isomorphism as much as possible: starts with simple patterns (paths), used to mine slightly more complex patterns (trees), then graphs.
  - The fastest of the four

Exercise: DFS search

Data:



Canonical codes



Different search paths may lead to the same pattern!

How to avoid exploring multiple times the same patterns?

- Have a generation strategy that limits duplicates
  - E.g. always expand from the latest expanded vertex (Mofa, gSpan, ...)
  - Does not suffice by itself: see image above
- Detect if a pattern can be found following another search path
  - Naive approach: compare with all generated patterns → infeasible in reasonable time and memory
  - **Canonical codes** (gSpan, FFSM, Gaston)

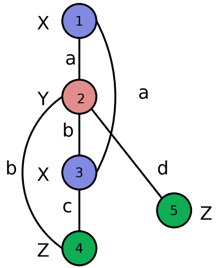
Canonical codes

1. Map each graph (2-dimensions) to a code (1-dimension) such that if two graph have equal codes they are isomorphic
2. Make codes comparable
   - The *minimum possible code* for a graph is called the **canonical code** of the graph[1]
   - Same canonical code $\iff$ isomorphic graphs
   - Canonical code uniquely identifies a graph
3. Only extend patterns on search paths that yield the canonical code for the pattern

[1] The maximum could also be used, it's arbitrary

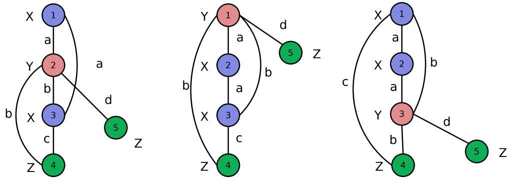
gSpan canonical code

- Code based on DFS construction of the graph (called DFS code)
- Each edge $e = (u, v)$ added to the graph is represented by a code element $(u, v, l(u), l(e), l(v))$



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

gSpan canonical code

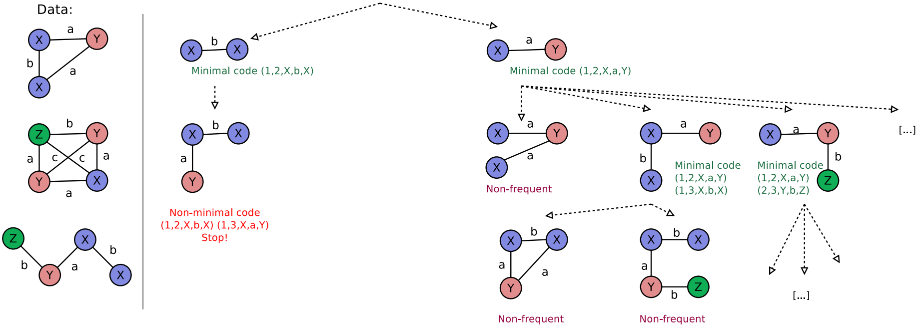


(a) (b) (c)

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

- Same graph can have different DFS codes depending on starting vertices
- Order defined on codes: lexicographic order of code elements
- When a pattern is generated during DFS search, decide if it could have a smaller DFS code. In that case, do not extend the pattern
  - It will be extended in the DFS branch where it has a minimal code
  - Assumes that the DFS search will eventually visit branches with minimal DFS code for any pattern

DFS pruning with canonical codes



Conclusion

- Graphs are a generic data structure that allows to express a large quantity of *structured* data
- However, graphs have additional complexity w.r.t. simpler data such as itemsets and sequential patterns, which can not be ignored when developing and using graph mining approaches
  - Pattern matching being a NP-complete subgraph isomorphism problem
  - Support computation
  - Recognizing if two graphs are the same (graph isomorphism)
  - ...
- Existing pattern mining approaches are constructed on the same basis as itemset mining (Apriori, pattern-growth), but need additional concepts to avoid too much complexity (e.g. canonical codes)

In pattern mining

The more generic the pattern/data language, the more it allows for expressiveness, but the more pattern mining tends to be difficult

Borgelt, C. and Berthold, M. R. (2002).
Mining molecular fragments: finding relevant substructures of molecules.
In *2002 IEEE International Conference on Data Mining, ICDM 2002. Proceedings.*, pages 51–58.

Bringmann, B. and Nijssen, S. (2008).
What Is Frequent in a Single Graph?
In *Advances in Knowledge Discovery and Data Mining*, volume 5012 of *Lecture Notes in Computer Science*, pages 858–863. Springer Berlin Heidelberg.

Gudes, E., Shimony, S. E., and Vanetik, N. (2006).
Discovering Frequent Graph Patterns Using Disjoint Paths.
IEEE Transactions on Knowledge and Data Engineering, 18(11):1441–1456.

Inokuchi, A., Washio, T., and Motoda, H. (2000).
An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data.
In *Principles of Data Mining and Knowledge Discovery*, volume 1910 of *Lecture Notes in Computer Science*, pages 13–23. Springer Berlin Heidelberg.

Kuramochi, M. and Karypis, G. (2001).

Frequent subgraph discovery.
In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 313–320.

Kuramochi, M. and Karypis, G. (2004).
Finding Frequent Patterns in a Large Sparse Graph.
In *Proceedings of the 2004 SIAM International Conference on Data Mining*, Proceedings, pages 345–356. Society for Industrial and Applied Mathematics.

Nijssen, S. and Kok, J. N. (2005).
The Gaston Tool for Frequent Subgraph Mining.
Electronic Notes in Theoretical Computer Science, 127(1):77–87.

Wang, W., Huan, J., and Prins, J. (2003).
Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism.
In *Third IEEE International Conference on Data Mining(ICDM)*, pages 549–552.

Wörlein, M., Meinl, T., Fischer, I., and Philippsen, M. (2005).
A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston.
In *Knowledge Discovery in Databases: PKDD 2005*, volume 3721 of *Lecture Notes in Computer Science*, pages 392–403. Springer Berlin Heidelberg.

Yan, X. and Han, J. (2002).
gSpan: Graph-based substructure pattern mining.
In *2002 IEEE International Conference on Data Mining. Proceedings*, pages 721–724. IEEE.