References

- [1] Davide Mottin, Anton Tstitsulin’s lectures (2017) – Hasso Plattner Institute
- [2] Slides of Francesco Bariatti (04/01/2021)

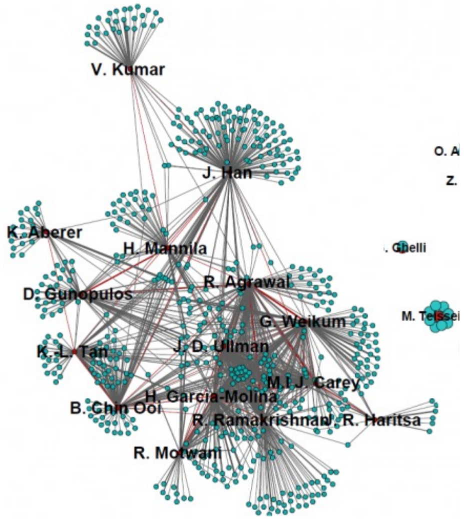
DMV

Graph Mining

Peggy Cellier – peggy.cellier@insa-rennes.fr
Département Informatique

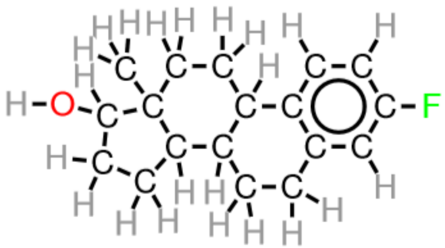
(last revision: october 2024)

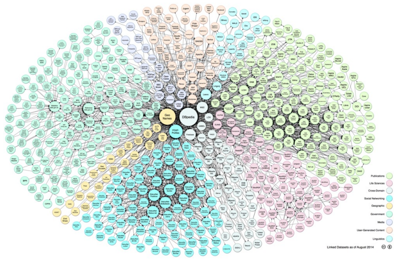
Motivation

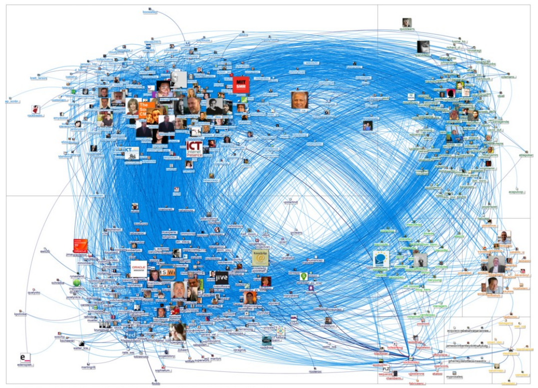


- **Huge quantity of graph data available**
  - Physical networks (telco,…)
  - Social networks
  - Molecules
  - Program call graph
  - Semantic web
  - ….

- **Interest to** **discover subgraph** **patterns in this data**







Plan

- **Graph definition and problem statement**
- **Support and frequent graph patterns**
- **Algorithms**
  - Pattern-merging algorithms (Apriori-based, BFS)
  - Pattern-growth algorithms (gSpan, etc)

Graph

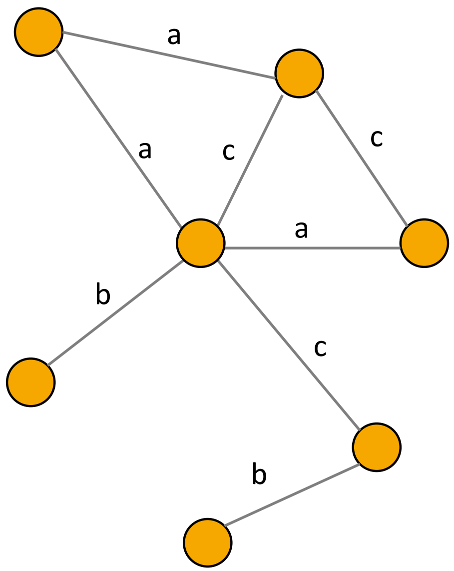
- **Graph definition**

$G = (V,E)$ $\quad E \subseteq V x V$

Set of vertices   Set of edges

- **Several kinds of graphs**
  - **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
  - Directed graphs
  - …

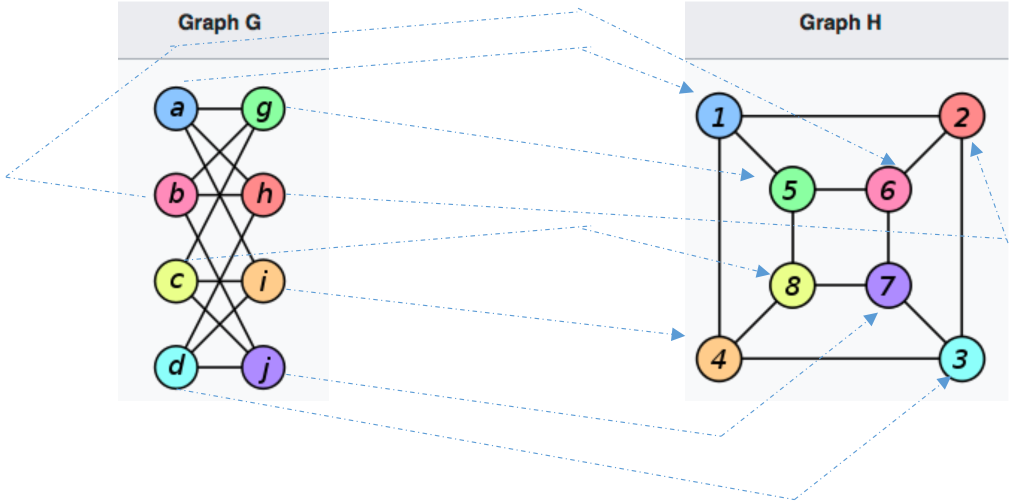
- **Example**
  - Undirected labeled graph with 7 vertices and 8 edges



- **Remarks**
  - Most graph mining approaches focus on undirected labeled graphs
  - Graphs are sometimes called networks depending on the domain

Look at those graphs

- **Do you think they have common parts?**



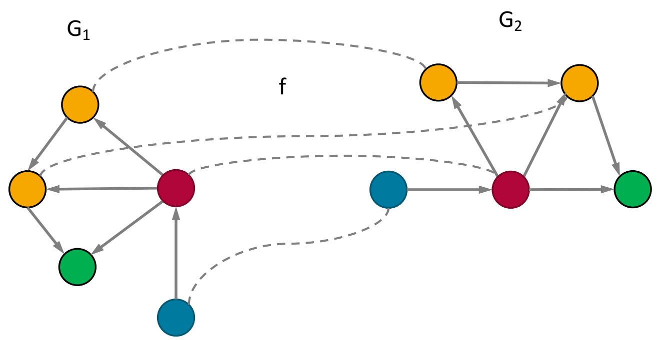
- **Graphs can appear different but actually have the same structure**

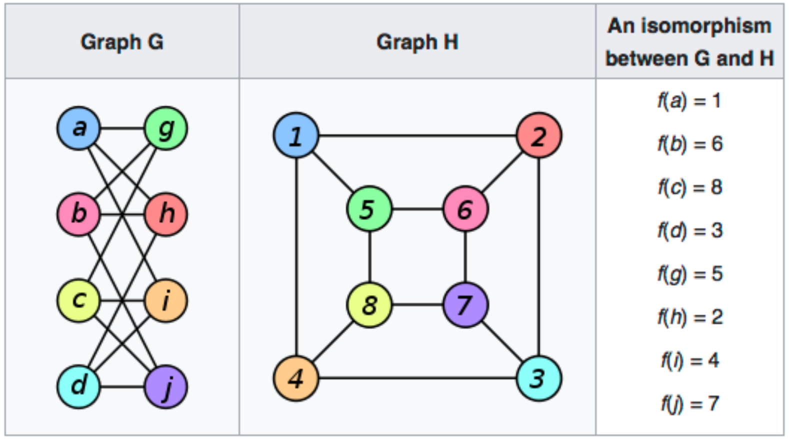
Graph isomorphism

- **Graph isomorphism:**
  - Recognizing if two graphs are the same

- **Graph Isomorphism**
  - Let us consider
    - G1=(V1,E1,l1)
    - G2=(V2,E2,l2)
  - G1 is isomorphic to G2 iff it exists a bijection function f: V1 -> V2 such that:
    - ∀ v1 ∈ V1 => l1(v1)=l2(f(v1)) // same label
    - (v1,u1) ∈ E1 => (f(v1),f(u1)) ∈ E2 // adjacent

- **Examples**



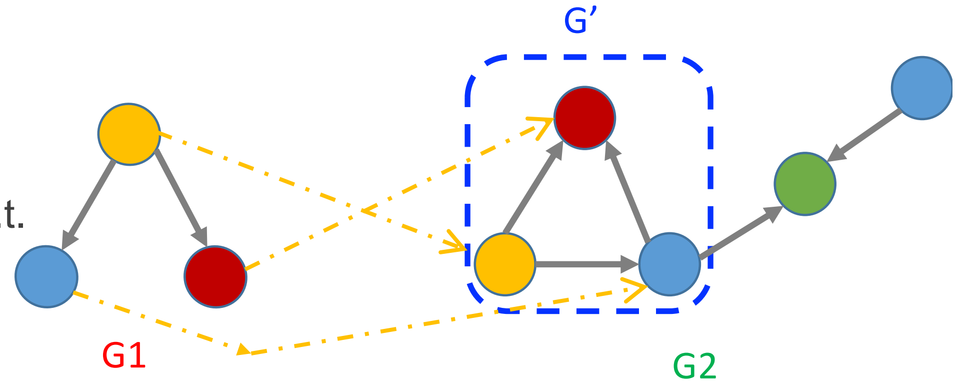


Source: https://en.wikipedia.org/wiki/Graph_isomorphism

Subgraph isomorphism

- **Subgraph isomorphism:**
  - Recognizing if a graph is a part of another graph

- **Subgraph isomorphic**
  - Let us consider two graphs: G2 and G1
  - G1 is subgraph isomorphic to G2 if there exists G' s.t.
    - G' is isomorphic to G1
    - G' is a subgraph of G2
  - More formally:
  - G1=(V1,E1,I1) is subgraph isomorphic to G2=(V2,E2,I2) if
    - there exists an injective function f: V1 -> V2 s.t.:
    - $\forall e = (u,v) \in E1 \quad (f(u),f(v)) \in E2$
    - $\forall v \in V1 \; I2(f(v)) = I1(v)$
    - $\forall e=(u,v) \in E1 \; I2((f(u),f(v))) = I1((u,v))$



- **Remark: Subgraph isomorphism search is NP-complete!**
  - In practice if labels are diverse enough, it can be computed in reasonnable time.

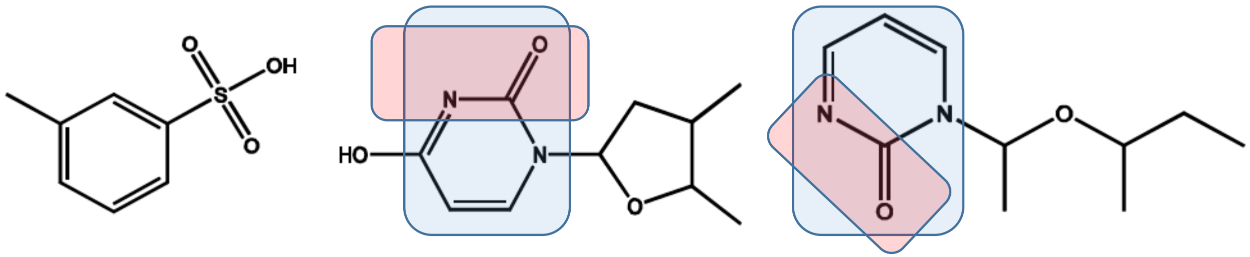
Graph Pattern Mining

- **Graph Mining is essentially the problem of discovering frequent subgraphs (patterns) occurring in the input data graph(s).**

- **Motivation**
  - 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

Graph Mining

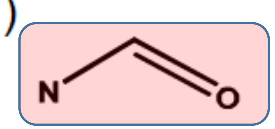
- **Example of subgraphs**
  - Graph dataset



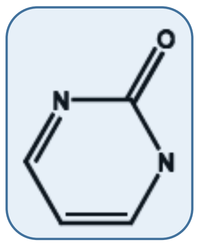
(A) (B) (C)

  - Example of frequent patterns

(1)



(2)



Plan

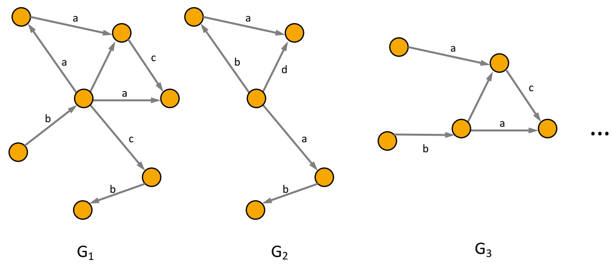
- **Graph definition and problem statement**
- **Support and frequent graph patterns**
- **Algorithms**
  - Pattern-merging algorithms (Apriori-based, BFS)
  - Pattern-growth algorithms (gSpan, etc)

What is frequent?

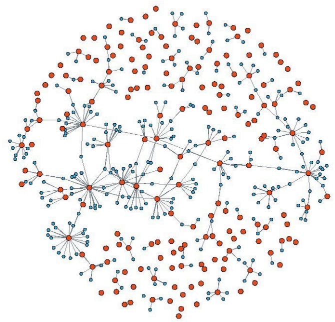
- **Discovering frequent subgraphs =** discovering subgraphs with support > minsup

- **Support definition depends on the kind of graph data**
  - Basic 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 (small) graphs
    - E.g. molecules, sentences

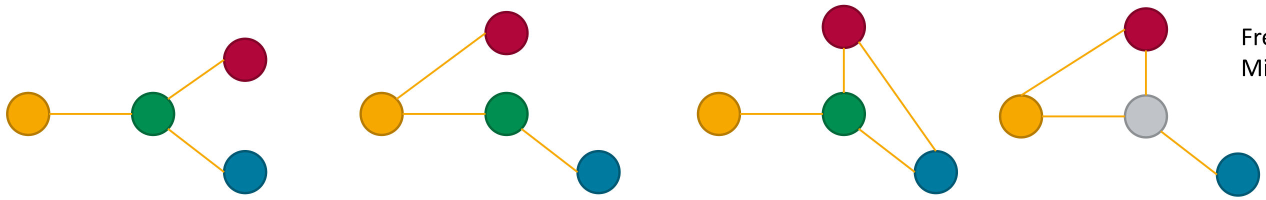


  - **Single graph**: the data is a unique (generally large) graph
    - E.g. semantic web, social networks, DNA



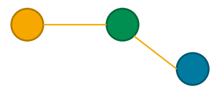
What is frequent in a graph collection?

- **Support definition**
  - Let D be a graph collection and P a graph pattern
  - Support(P) = $\frac{|\{g \in D \mid P \text{ is a subgraph isomorphic to } g\}|}{|D|}$
- **Remark**
  - Each graph of the collection can only contribute once to the support, even if it has multiple occrurrences of the pattern
- **Example**



G1 G2 G3 G4

Frequent subgraph
Min support = 3/4

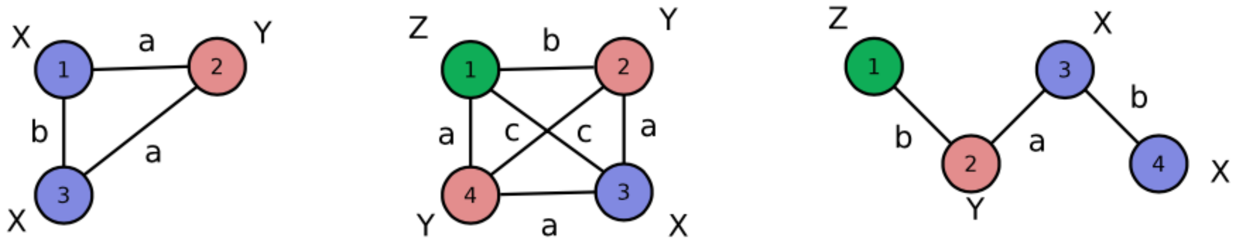


- **Support is anti-monotonic**
  - The support of a graph is lower or equal to support of its subgraphs

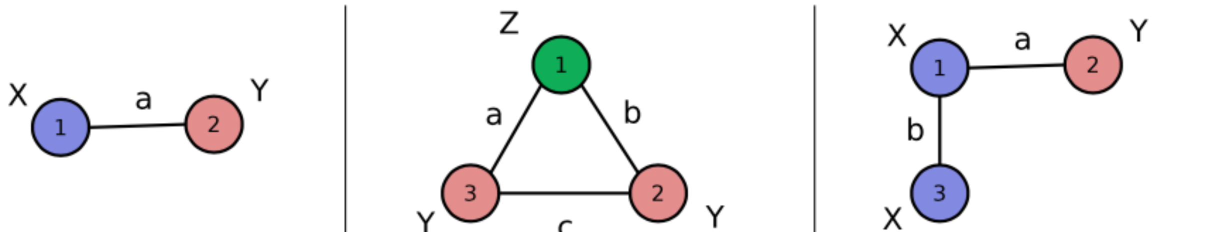
What is frequent in a graph collection?

- **Exercise: Compute the support of those patterns**

Data:



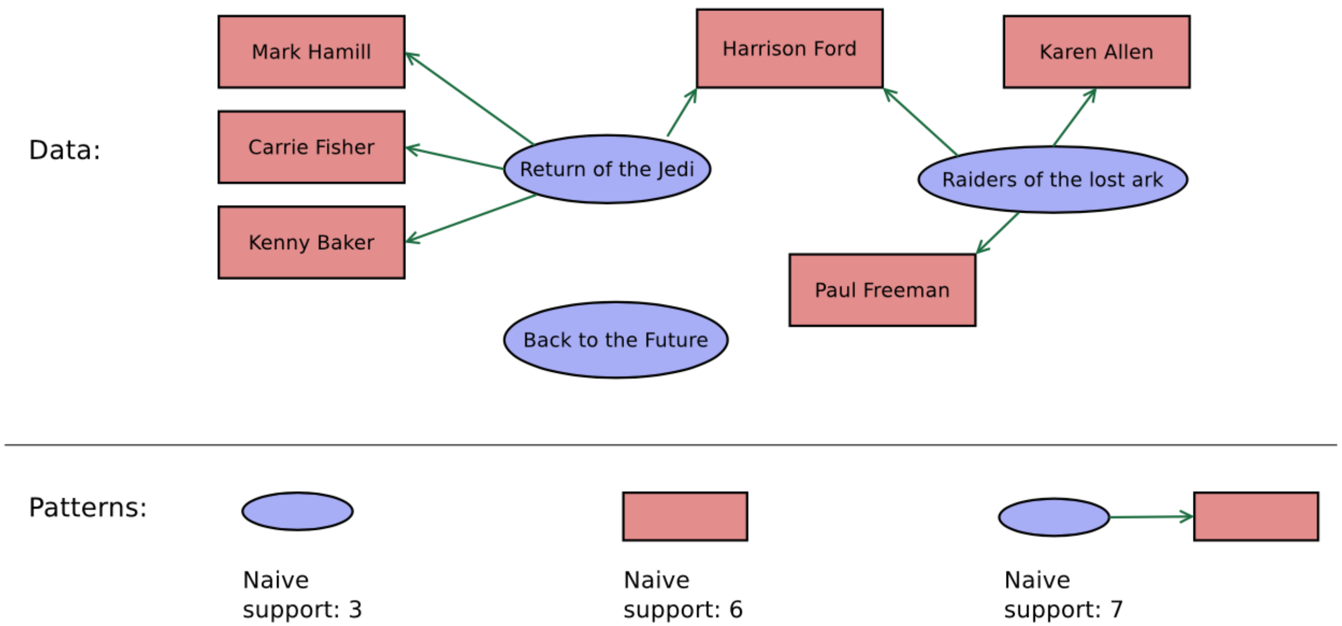
Patterns:



Support: Support: Support:

What is frequent in a single graph?

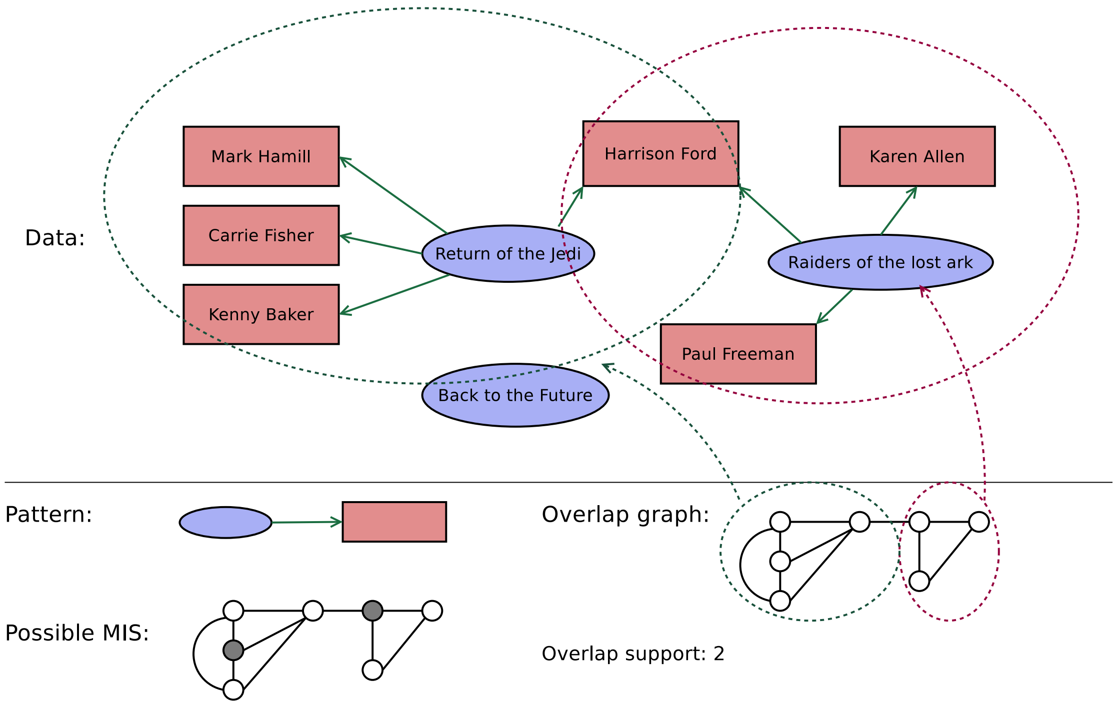
- **Naive solution**
  - Count how many occurrences the pattern has in the graph



Naive support is **not** anti-monotonic

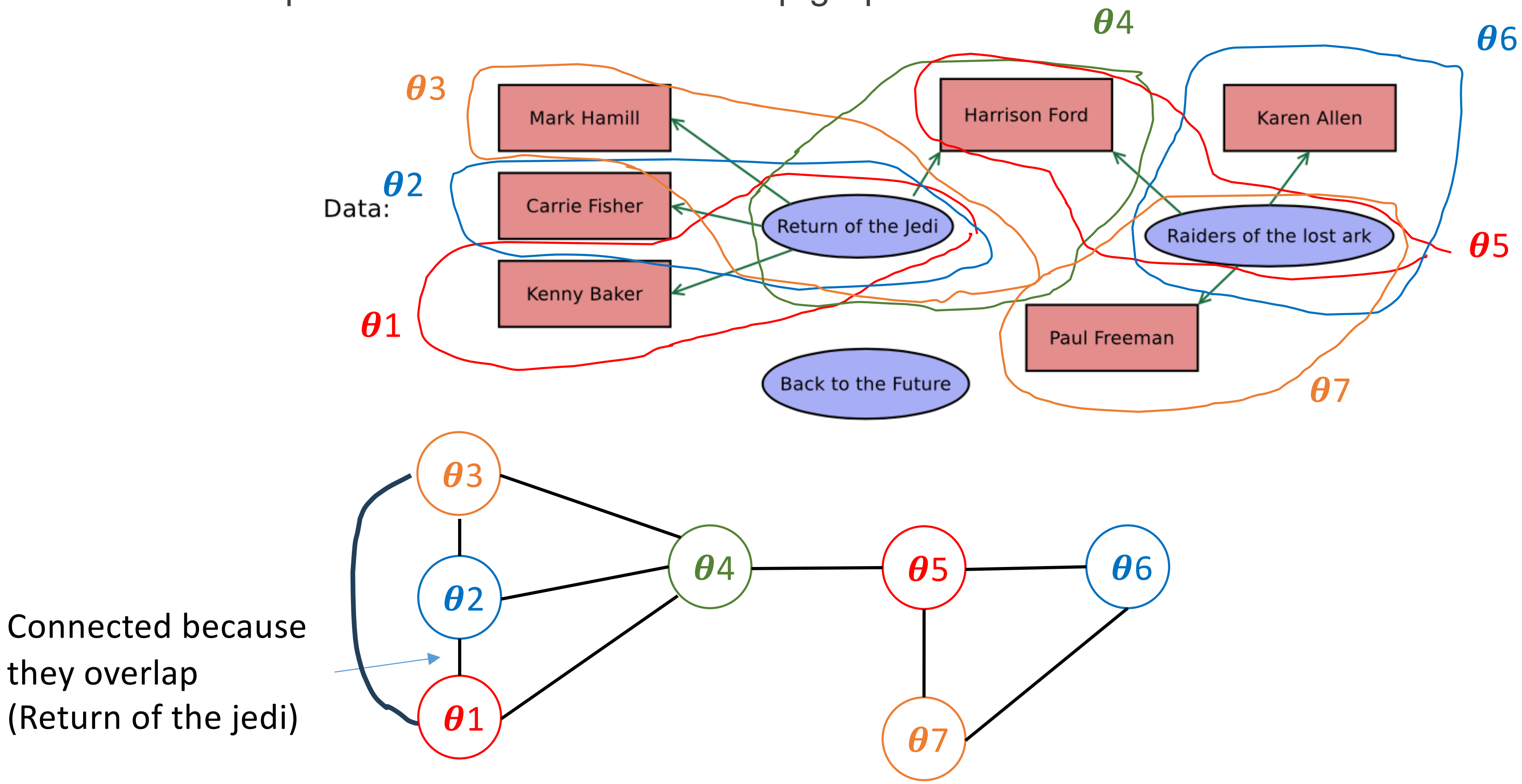
What is frequent in a single graph?

- **Overlap-based approaches** [Kuramochi and Karypis, 2004]
  - Compute overlap graph of pattern embeddings



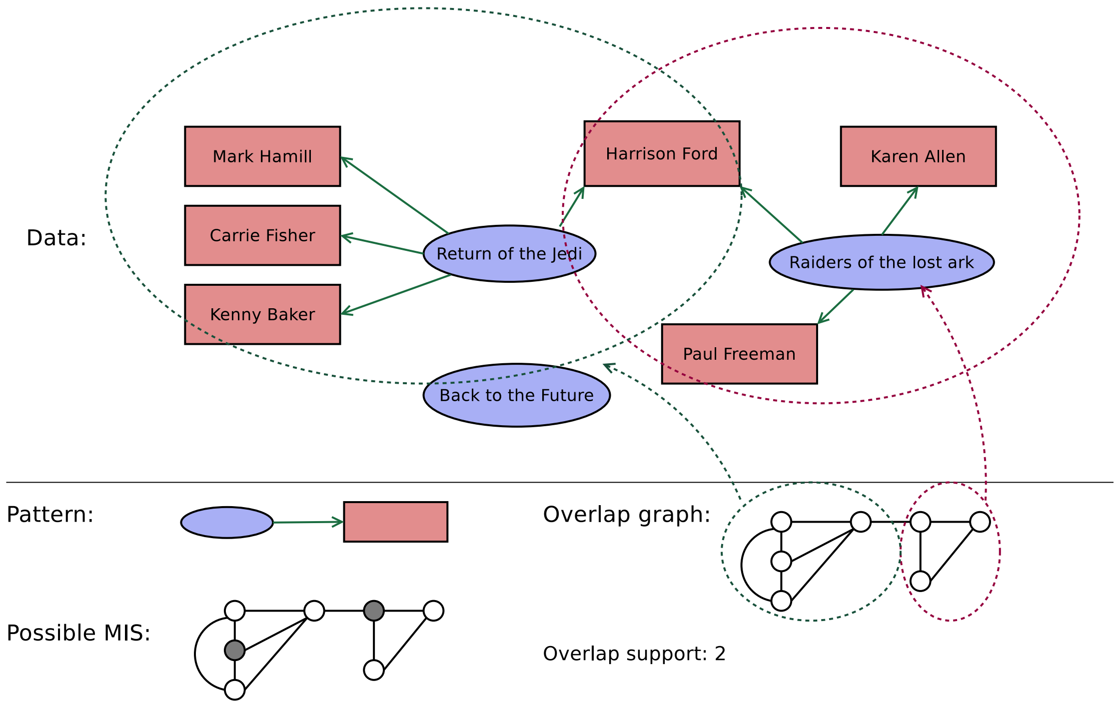
What is frequent in a single graph?

- **Overlap-based approaches** [Kuramochi and Karypis, 2004]
  - Example: construction of the overlap graph



What is frequent in a single graph?

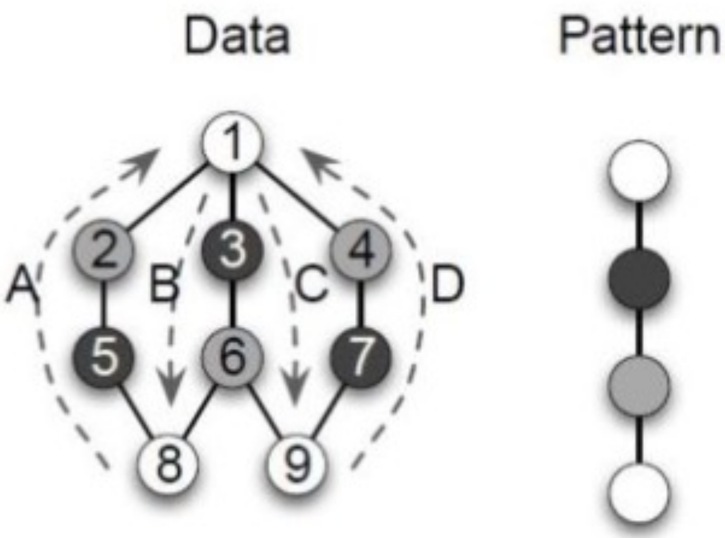
- **Overlap-based approaches** [Kuramochi and Karypis, 2004]
  - Compute overlap graph of pattern embeddings
  - Support is the size of MIS (**Maximum Independant Set**) of overlap graph
    - i.e. maximum number of non-overlapping embeddings of the patterns



- 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 D be a single graph and P a graph pattern
  - Support(P) = $\min_{v \in V^P} |\{\boldsymbol{\varepsilon}(v) \mid \boldsymbol{\varepsilon} \textit{ is an embedding of } P \textit{ in } D\}|$



4

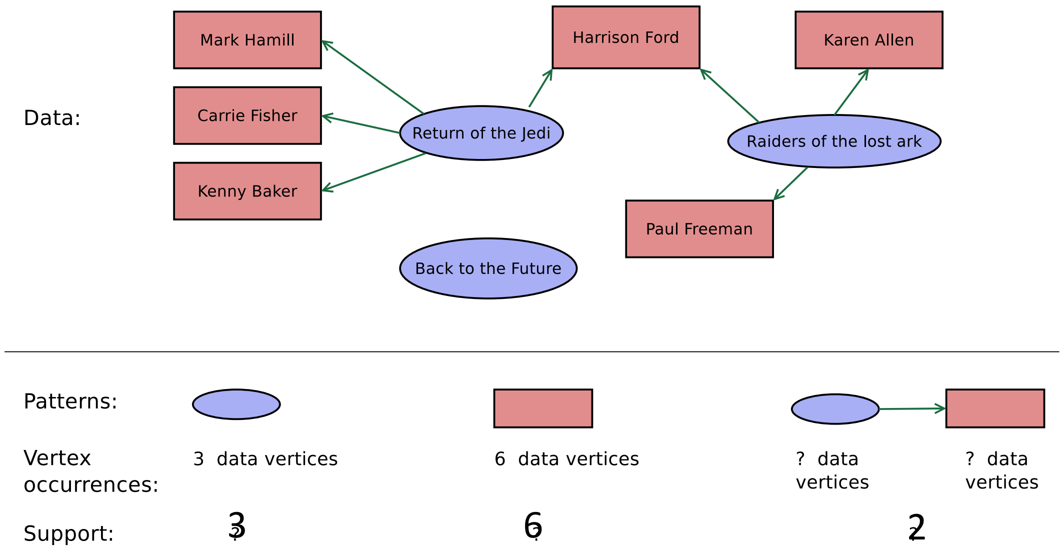
Embeddings

A	B	C	D	
8	1	1	9	⇒ 3 different values
5	3	3	7	⇒ 3 different values
2	6	6	4	⇒ 3 different values
1	8	9	1	⇒ 3 different values

⇒ **Support = 3**

What is frequent in a single graph?

- **Minimum image based support** [Bringmann and Nijssen, 2008]
  - Let D be a single graph and P a graph pattern
  - Support(P) = $\min_{v \in V^P} |\{\boldsymbol{\varepsilon}(v) \mid \boldsymbol{\varepsilon} \; is\ an\ embedding\ of\ P\ in\ D\ \}|$



  - Minimum image based support is anti-monotonic
  - And it does not need to compute a NP-complete problem

Plan

- **Graph definition and problem statement**
- **Support and frequent graph patterns**
- **Algorithms**
  - **Pattern-merging algorithms (Apriori-based, BFS)**
  - Pattern-growth algorithms (gSpan, etc)

Pattern-merging algorithms

- **Based on the property:**
  - If the support measure is anti-monotonic
  - For a k-pattern to be frequent
  - All (k-1)-patterns contained in the k-pattern must be frequent.

- **Work simirlaly 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

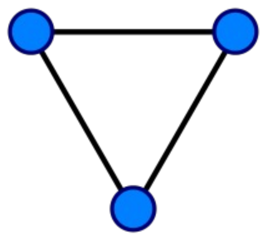
Pattern-merging algorithms

- **Main pattern-merging graph mining algorithms**
  - AGM/AcGM [Inokuchi et al., 2000]
  - FSG [Kuramochi and Karypis, 2001]
  - FFSM [Huan, et al., 2003] and SPIN [Huan et al., 04]
  - DPMine [Gudes et al., 2006]

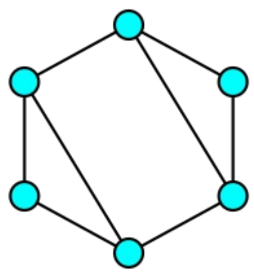
- **Main differences is the definition of a k-pattern**
  - K vertices
  - K edges
  - K edge-disjoint paths
  - …

FSG: k-subgraph

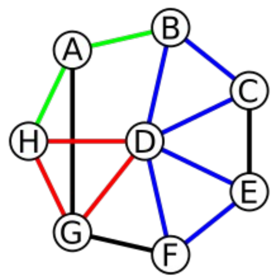
- **Notation:** k-subgraph is a subgraph with k edges



?-subgraph



?-subgraph



?-subgraph

FSG

- **Init:**
  - Scan the transactions to find $\mathcal{F}_1$;
    - $\mathcal{F}_1$ = set of all frequent 1-subgraphs and 2-subgraphs, together with their counts

For (k=3; $\mathcal{F}_{k-1} \neq \emptyset$; k++)

1. **Candidate Generation** – C_k, the set of candidate k-subgraphs, from $\mathcal{F}_{k-1}$, the set of frequent (k-1)-subgraphs;

2. **Candidates pruning** - a necessary condition of candidate to be frequent is that each of its (k-1)-subgraphs is frequent.

3. **Frequency counting** - Scan the graph database to count the occurrences of subgraphs in C_k;

4. $\mathcal{F}_k = \{c \in C_K \mid c \text{ has counts} \geq min\_sup\}$

Return $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \ldots\ldots \cup \mathcal{F}_k$ (= $\mathcal{F}$)

Pattern-merging algorithms: Simple operations?

- **Candidate generation**
  - To determine two candidates for joining, we need to check for graph isomorphism.
- **Candidate pruning**
  - To check downward closure property, we need graph isomorphism.
- **Frequency counting**
  - Subgraph isomorphism for checking containment of a frequent subgraph.

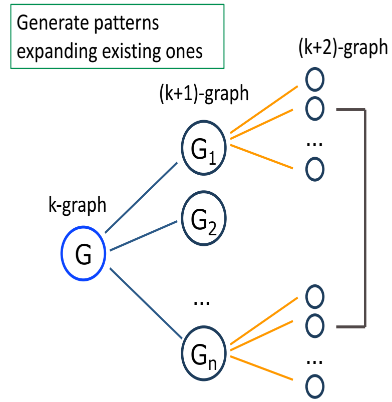
Recall that subgraph isomorphism is **NP**-complete!!!

Plan

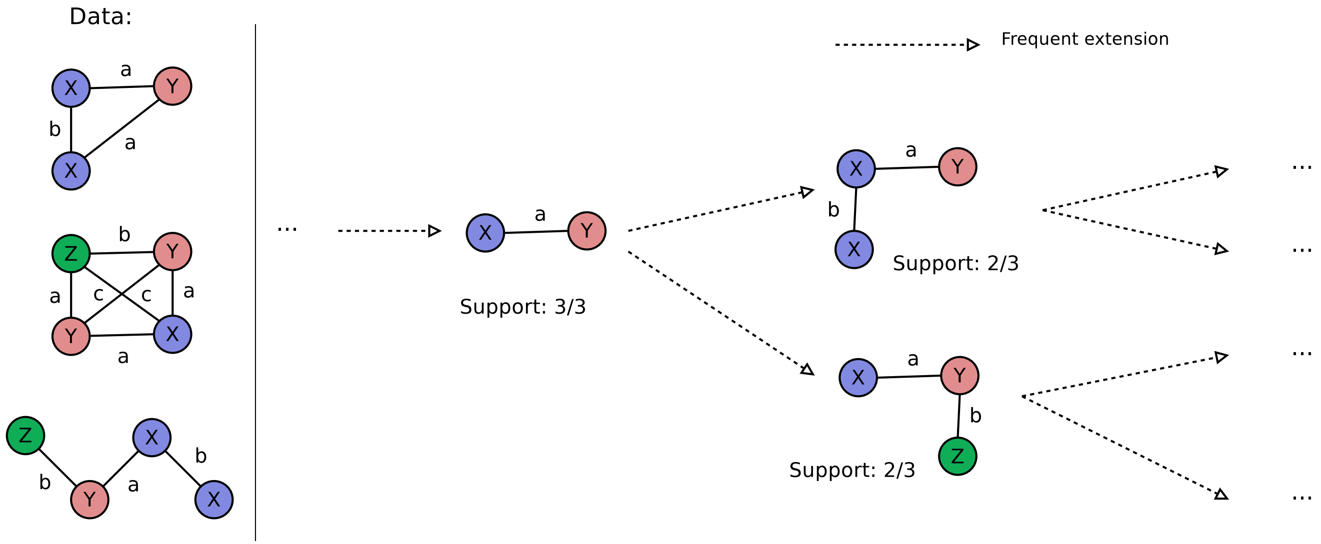
- **Graph definition and problem statement**
- **Support and frequent graph patterns**
- **Algorithms**
  - Pattern-merging algorithms (Apriori-based, BFS)
  - **Pattern-growth algorithms (gSpan, etc)**

Pattern growth approach

- **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-patterns to generate (k+1)-patterns => memory gain
    - Only generate frequent patterns => avoid testing non-frequent candidates => time gain

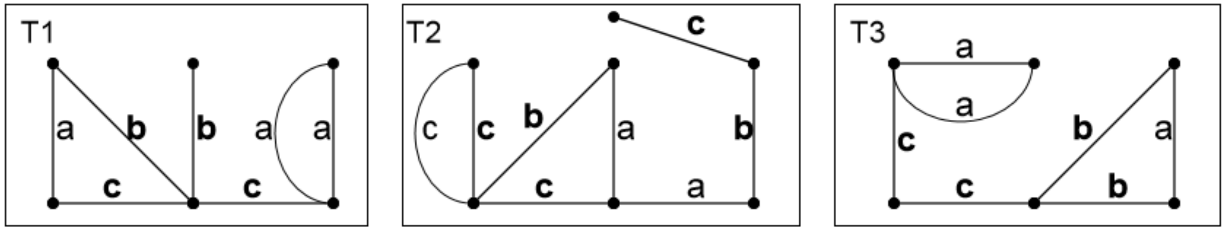


- **Most algorithms in this family use depth-first search to generate patterns**
  - Often called DFS algorithms

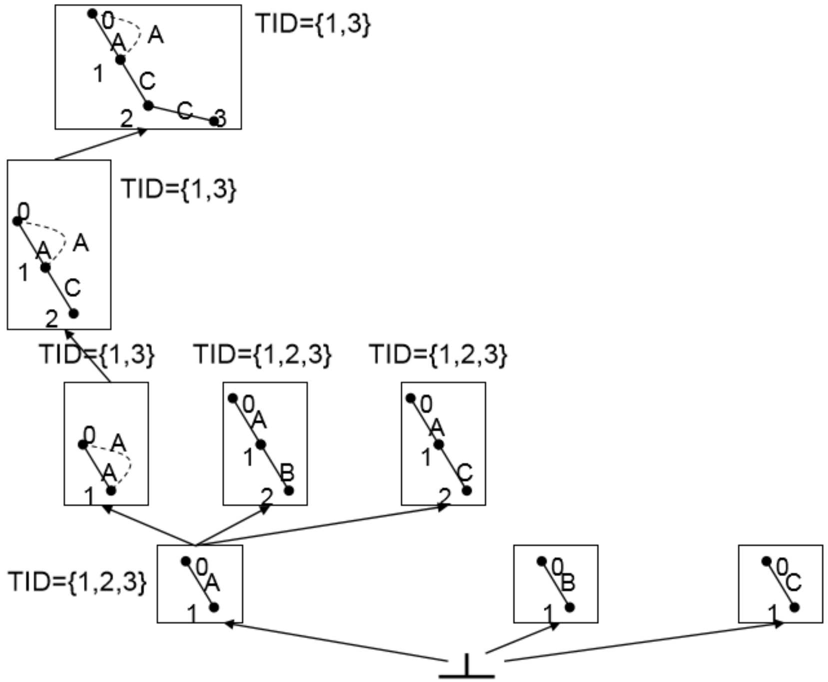


Example

- **Given a database**



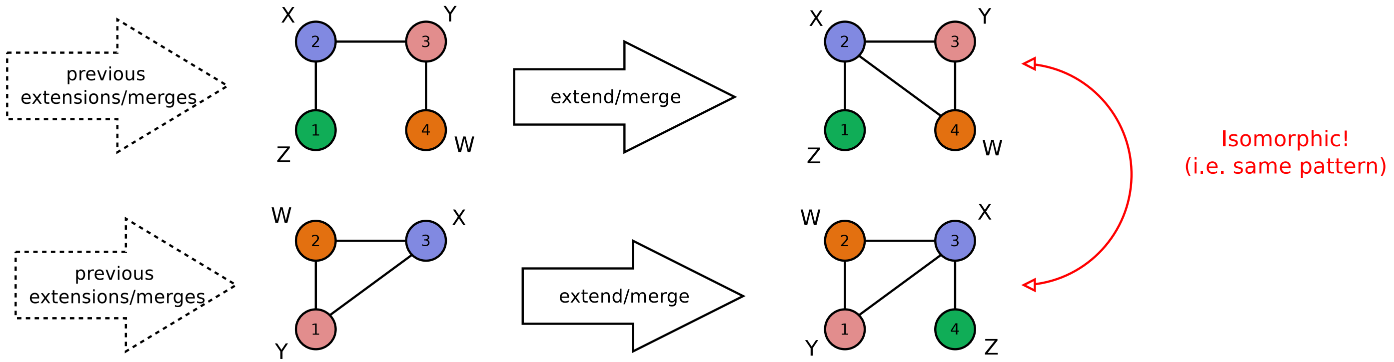
- **Example of first steps**



Pattern-growth algorithms

- **Most 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
      - Depth First Search (DFS) with rightmost path expansion
  - FFSM [Wang et al., 2003]
    - Uses both pattern expansion 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

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 (cf example above)
  - Detect if a pattern can be found following another search path
    - Naive approach: compare with all generated patterns => not possible in reasonnable time and memory
    - **Canonical codes** (gSpan, FFSM, Gaston)

Canonical Codes

- **Idea**
  - Map each graph (2-dimensions) to a code (1-dimension) such that if two graphs have equal codes they are isomorphic

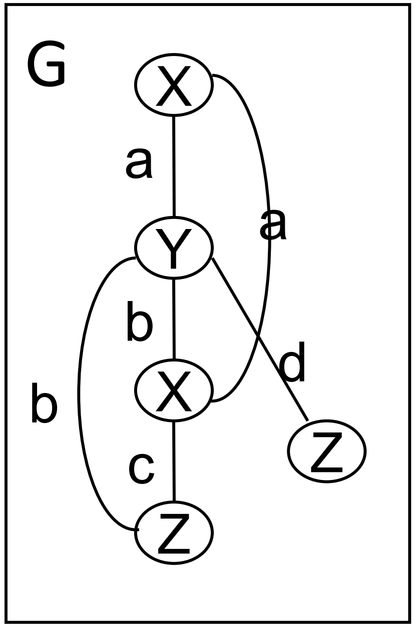
- **Make code comparable**
  - The **minimum possible code** for a graph is called the canonical code of the graph.
  - Same canonical code ⇔ isomorphic graphs
  - Canonical code uniquely identifies a graph

- **In the generation**
  - Only extend patterns on search paths that yield the canonical code for the pattern

Note: The maximum code could be used instead of the minimal, it's arbitrary.

gSpan Canonical Code (DFS code)

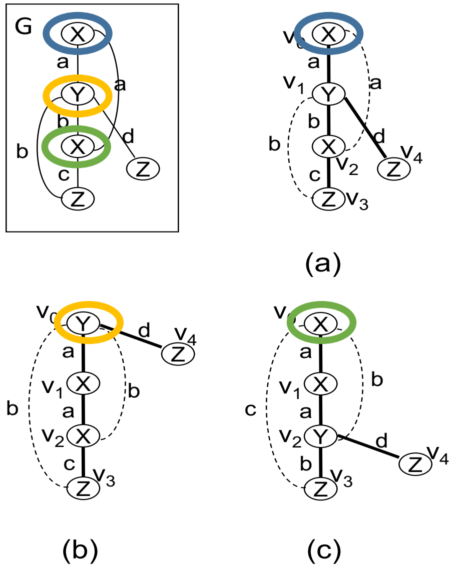
- **Code based on DFS construction of the graph (called DFS code)**
- **Each edge e=(u,v) added to the graph is represented by a code element**
  - (u,v,l(u), l(e), l(v))



Code
(0, 1, X, a, Y)
(1, 2, Y, b, X)
(2, 0, X, a, X)
(2, 3, X, c, Z)
(3, 1, Z, b, Y)
(1, 4, Y, d, Z)

Single graph → several DFS-codes

	(a)	(b)	(c)
1	(0, 1, X, a, Y)	(0, 1, Y, a, X)	(0, 1, X, a, X)
2	(1, 2, Y, b, X)	(1, 2, X, a, X)	(1, 2, X, a, Y)
3	(2, 0, X, a, X)	(2, 0, X, b, Y)	(2, 0, Y, b, X)
4	(2, 3, X, c, Z)	(2, 3, X, c, Z)	(2, 3, Y, b, Z)
5	(3, 1, Z, b, Y)	(3, 0, Z, b, Y)	(3, 0, Z, c, X)
6	(1, 4, Y, d, Z)	(0, 4, Y, d, Z)	(2, 4, Y, d, Z)

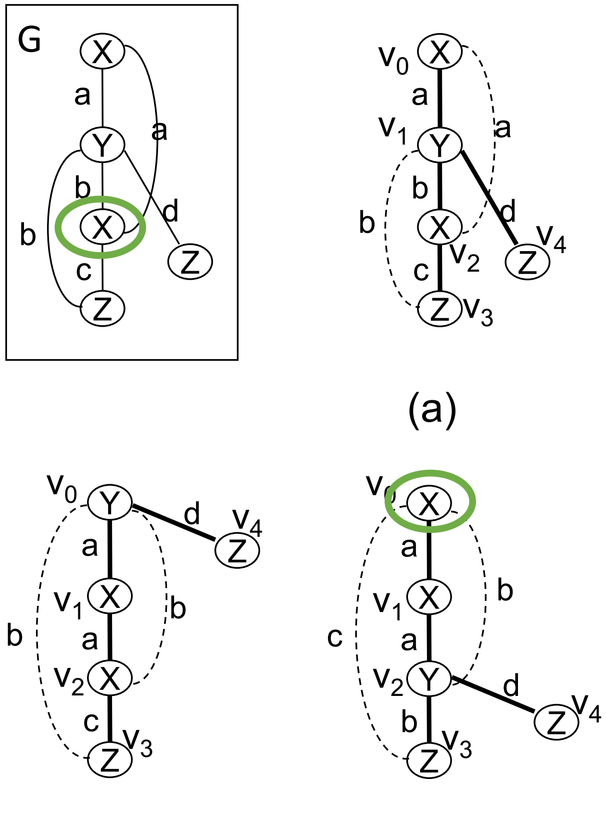


- **Same graph can have different DFS codes depending on starting vertices**
- **Order defined on codes**
  - Lexicographic order of code elements
- **When a pattern is generated during DFS search, decide if it could have a smaller DFS code**
  - Yes => do not extend
  - No => extend the DFS branch where it has a minimal code

Single graph → single Min DFS-code

Min DFS-Code

	(a)	(b)	(c)
1	(0, 1, X, a, Y)	(0, 1, Y, a, X)	**(0, 1, X, a, X)**
2	(1, 2, Y, b, X)	(1, 2, X, a, X)	**(1, 2, X, a, Y)**
3	(2, 0, X, a, X)	(2, 0, X, b, Y)	**(2, 0, Y, b, X)**
4	(2, 3, X, c, Z)	(2, 3, X, c, Z)	**(2, 3, Y, b, Z)**
5	(3, 1, Z, b, Y)	(3, 0, Z, b, Y)	**(3, 0, Z, c, X)**
6	(1, 4, Y, d, Z)	(0, 4, Y, d, Z)	**(2, 4, Y, d, Z)**

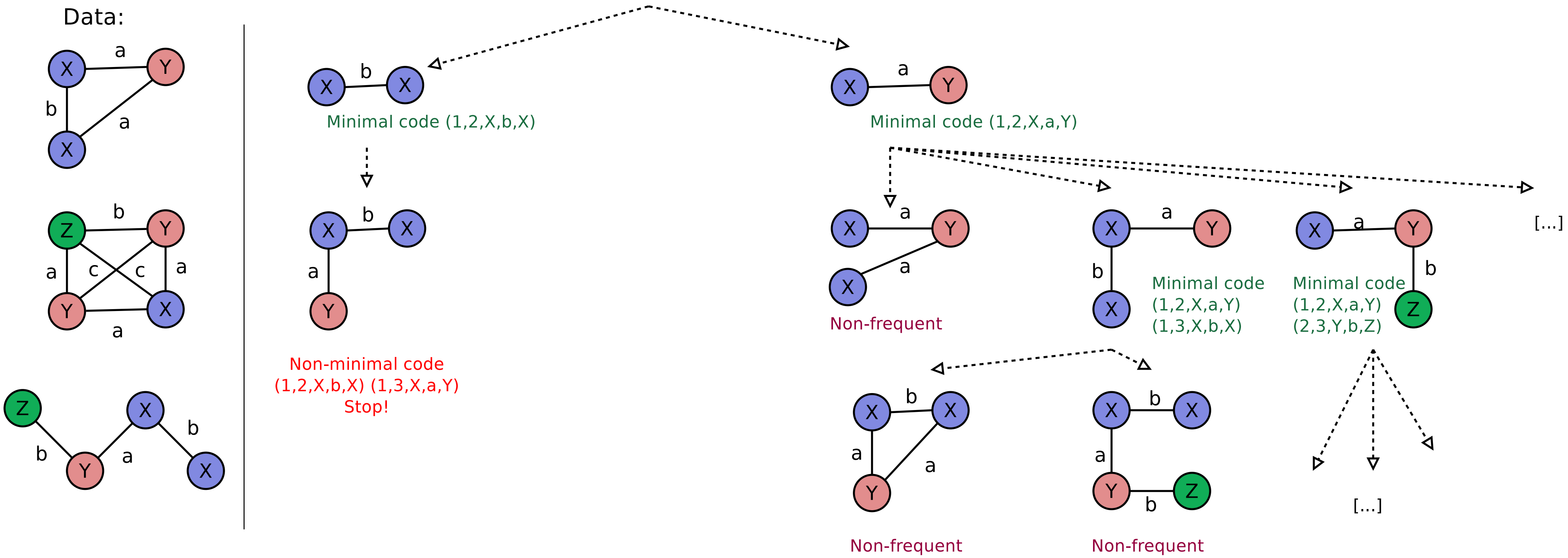


(a)

(b)

(c)

DFS Pruning with Canonical Codes



Conclusion

- **Graphs are a generic data structure that allows to express a large quantity of structured data**

- **However, graphs more complexe than itemsets or sequences**
  - Pattern matching is a NP-complete subgraph isomorphism problem
  - Support computation
  - Similarity between two graphs
  - …

- **Graph 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 the pattern mining tends to be difficult