October 31, 2025

Inter-University Research Institute Corporation /
Research Organization of Information and Systems
National Institute of Informatics

Graph Mining

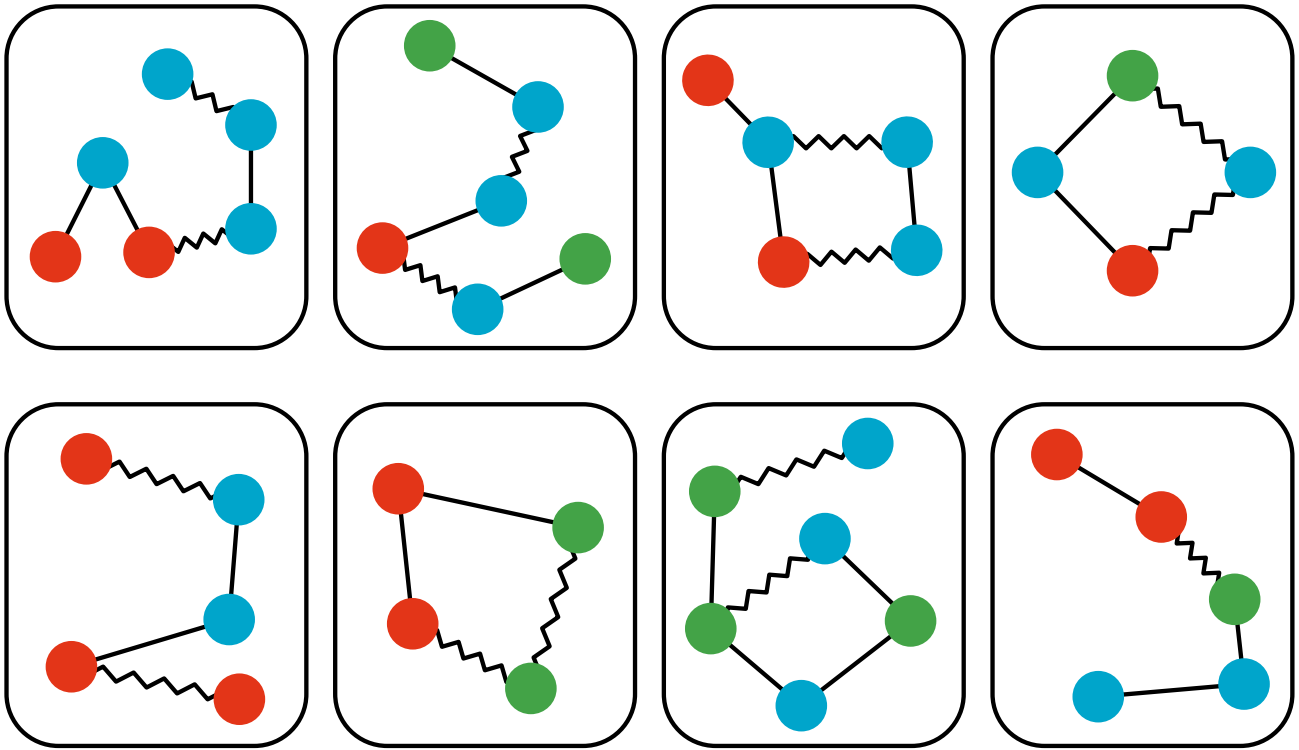
Data Mining 03 (データマイニング)

Mahito Sugiyama (杉山麿人)

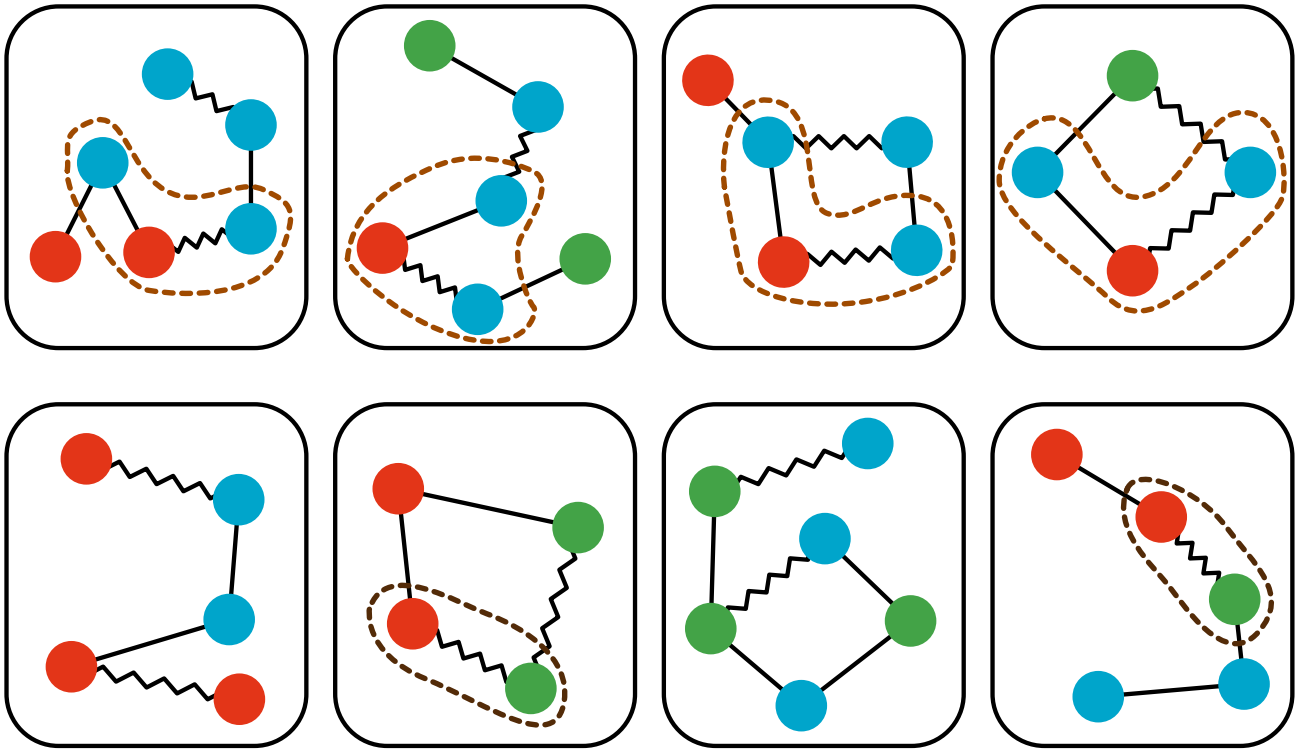
Today's Outline

- A primer of graphs
  - Subgraph isomorphism
- Graph mining
  - How to find (sub)graphs from graph databases?
  - Revisiting the Apriori principle to avoid combinatorial explosion
  - The canonical DFS code for graph representation

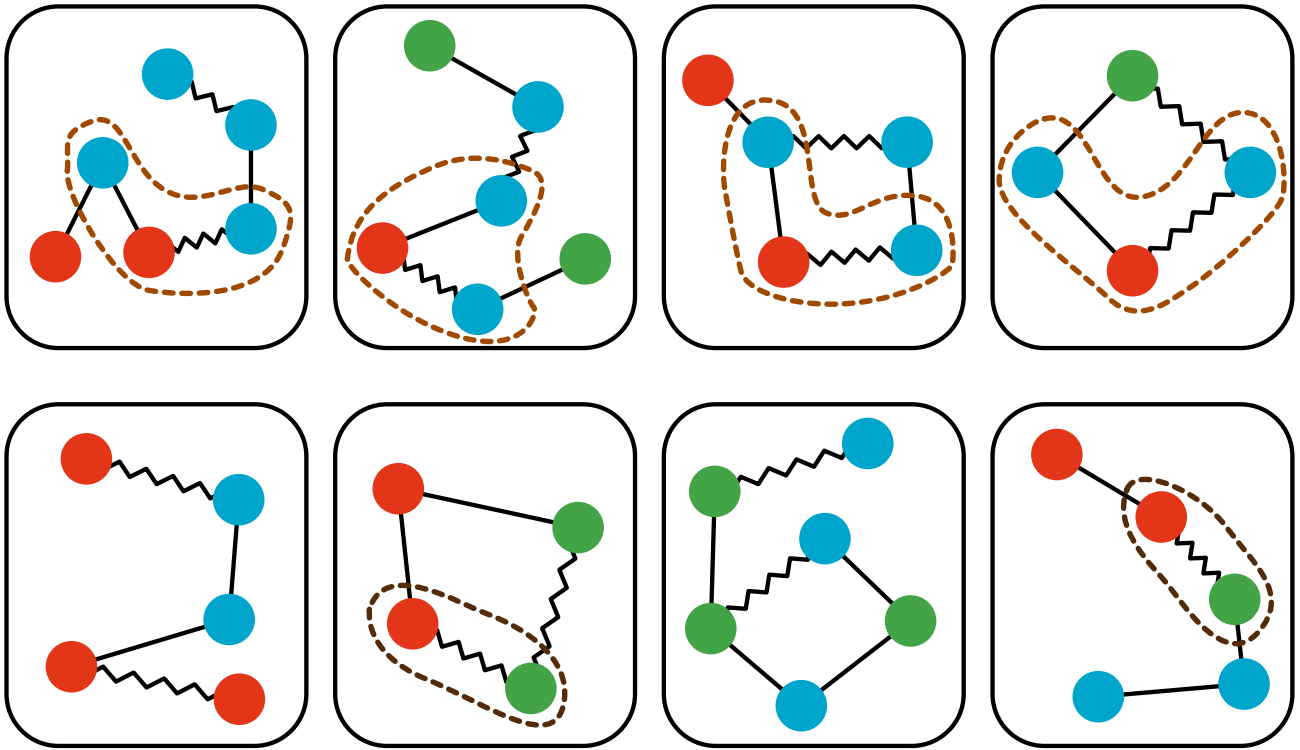
Graph Mining: Overview



Graph Mining: Overview



Graph Mining: Overview



Graphs

- An (unlabeled) graph $G = (V, E)$
  - V: a vertex (node) set, $E \subseteq V \times V$: an edge set
  - For $(u, v) \in E$, vertices u, v are adjacent, v is a neighbor of u
    - We do not distinguish (u, v) and (v, u) if the graph is undirected
  - The set of all neighbors of v is denoted as

    $N(v) = \{u \in V \mid (v, u) \in E\}$
    - It may be denoted as $N_G(v)$ to explicitly indicate it is defined w.r.t. G
- A labeled graph $G = (V, E, \phi)$
  - $\phi : V \cup E \to \Sigma$ gives labels (attributes) to V or E
    - Σ is the set of vertex and edge labels

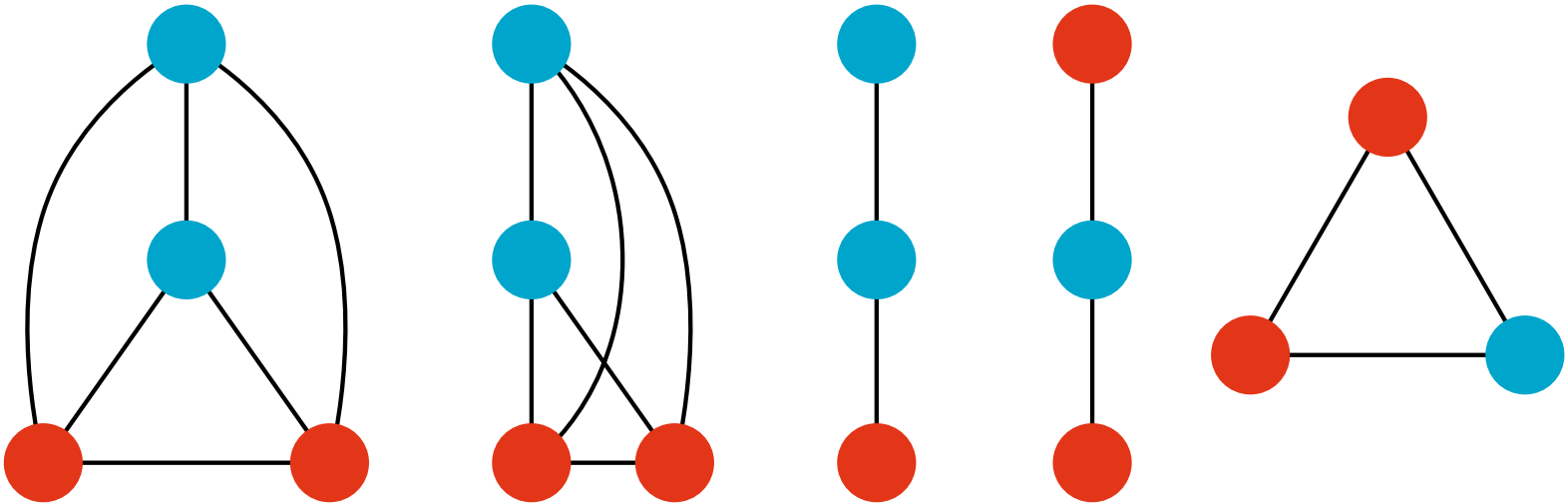
Subgraph Isomorphism

- For any subset $S \subseteq V$ of $G = (V, E)$, its (induced) subgraph is $G[S] = (S, T)$ with $T = \{(u, v) \in E \mid u, v \in S\}$
- A graph $H = (S, T)$ is isomorphic to G if there exists a bijective function $\pi : S \to V$ such that

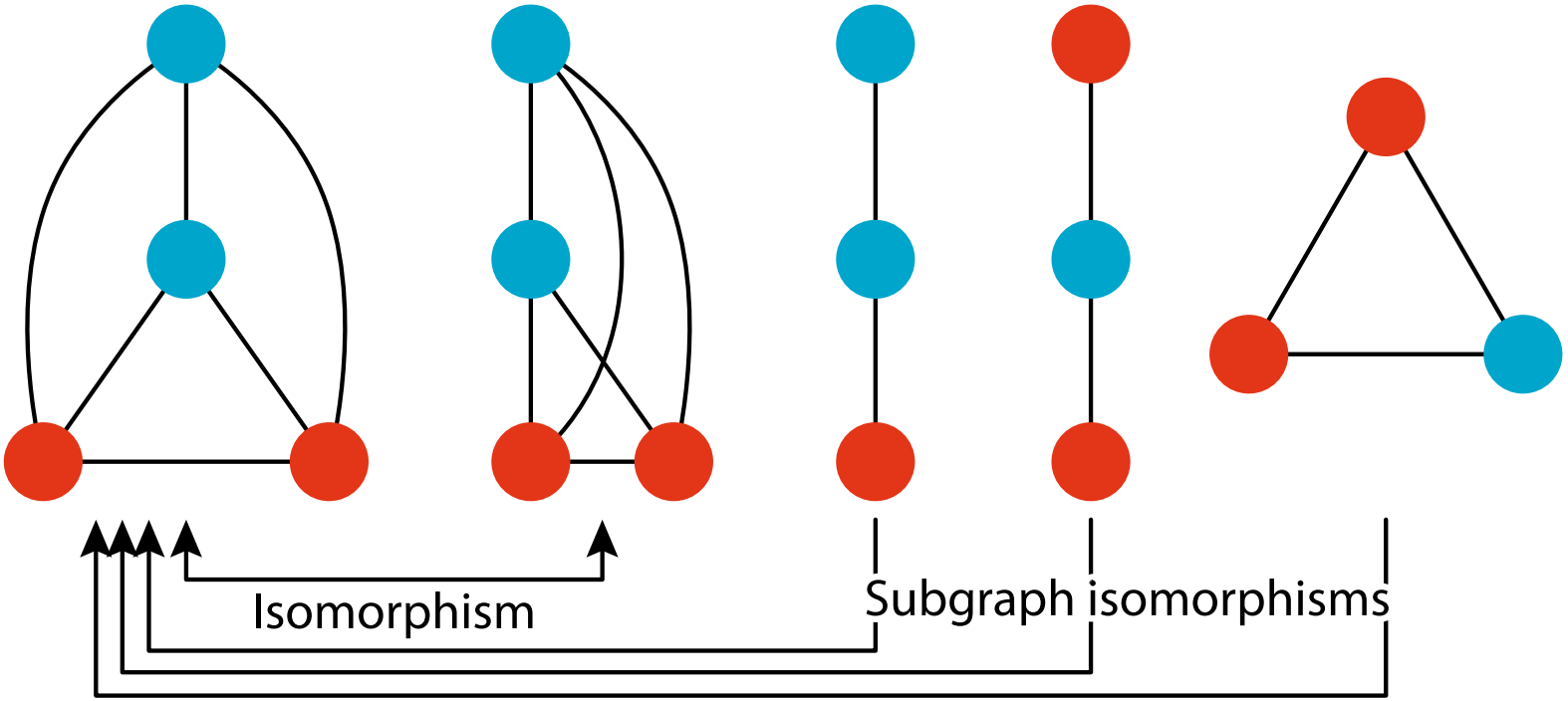
  (i) $(u, v) \in T \iff (\pi(u), \pi(v)) \in E$

  (ii) $\forall v \in S,\ \phi(v) = \phi(\pi(v))$ **(iii)** $\forall (u, v) \in T,\ \phi(u, v) = \phi(\pi(u), \pi(v))$
- If π is injective but not surjective: $G \setminus \text{range}(\pi) \neq \emptyset$, H is subgraph isomorphic to G, denoted by $H \sqsubseteq G$
  - Testing whether $H \sqsubseteq G$ is NP-complete (computationally heavy!)

Subgraph Isomorphism Examples



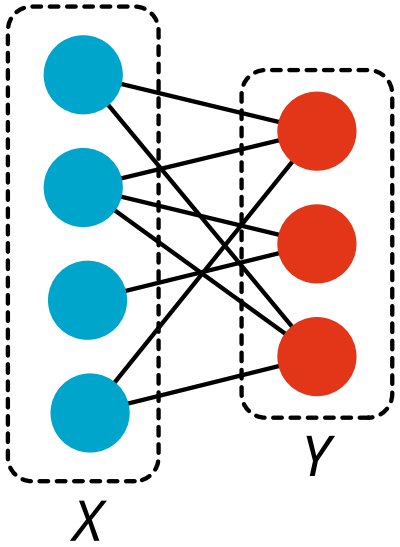
Subgraph Isomorphism Examples



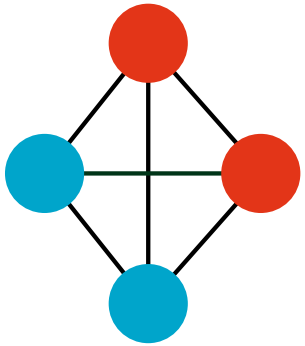
Frequently Used Definitions in Graphs

- A bipartite graph $G = (V, E)$: V can be divided into two independent sets X and Y such that $X \cup Y = V$ and $X \cap Y = \emptyset$
  - An independent set $X \subseteq V$ of G: $(u, v) \notin E$ for any $u, v \in X$
- A clique $C \subseteq V$ in G: $(u, v) \in E$ for every $u, v \in C$
- A tree: An undirected graph where every pair of vertices is connected by exactly one path
  - A path: A sequence of vertices such that there is an edge between each vertex and its next vertex
- A spanning tree of G: A subgraph $H = (S, T)$ such that $S = V$ and H is a tree (always exists for connected graphs)

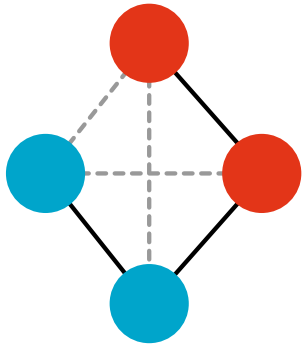
Examples



Bipartite graph



Clique



Spanning tree

Subgraph Mining

- In graph mining, pattern $\iff$ (sub)graph
- S: the set of graphs (can be infinite), a dataset D is a multiset of S
  - D is a collection of graphs: $D = \{G_1, G_2, \dots, G_n\}$
- The frequency $\eta(G)$ of a graph G is given as

$$\eta(G) = \frac{|\{G_i \in D \mid G \sqsubseteq G_i\}|}{|D|} = \frac{1}{|D|} \sum_{H \sqsupseteq G} \mathbf{1}_D(H)$$

Frequent subgraph mining problem:
Given a threshold σ,
enumerate the set $F = \{G \in S \mid \eta(G) \geq \sigma\}$

Two Challenges in Graph Mining

1. Combinatorial explosion of the search space
   - More massive than itemset mining
   - The number of subgraphs with m vertices: $O(2^{m^2})$
     - $O(m^2)$ possible edges
   - The number of subgraphs with m vertices and s labels: $O(s^{m^2})$
2. Subgraph isomorphism checking (does not exist in itemset mining)
   - When we obtain a subgraph H, computing $\eta(H)$ is heavy as we need to repeat subgraph isomorphism checking for every $G_i \in D$

- **Solution:** Use the Apriori principle and the (canonical) DFS code

Graph Mining Algorithms

- The first algorithm that achieves graph mining is AGM
  - Inokuchi, A., Washio, T., Motoda, H., **An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data**, *PKDD2000*
- The standard method is gSpan
  - Yan, X. & Han, J., **gSpan: Graph-based substructure pattern mining**, *ICDM2002*
- The state-of-the-art is GASTON
  - Nijssen, S. & Kok, J. N., **A Quickstart in Frequent Structure Mining Can Make a Difference**, *SIGKDD2004*

DFS Code (1/3)

- The DFS code represents a graph G as a sequence of tuples based on depth first search (DFS)
  - There can be multiple DFS codes for a single graph
- Perform DFS on a graph G and index each vertex by its discovery order
- Define two types of edges:
  Forward edges: Edges included in the DFS
  Backward edges: Other edges
- Each edge (i, j) is represented as a tuple $(i, j, \phi(i), \phi(j), \phi(i, j))$
  - $i < j$ if it is a forward edge and $i > j$ if backward

DFS Code (2/3)

- Introduce the (total) order "$<_t$" between two tuples t_1 and t_2 with $t_k = (i_k, j_k, \phi(i_k), \phi(j_k), \phi(i_k, j_k))$
- For edges, define $e_1 = (i_1, j_1) <_e e_2 = (i_2, j_2) \iff$

$j_1 < j_2$ or $(j_1 = j_2 \ \& \ i_1 > i_2)$	if both e_1 and e_2 are forward,
$i_1 < i_2$ or $(i_1 = i_2 \& j_1 < j_2)$	if both e_1 and e_2 are backward,
$j_1 \leq i_2$	if e_1 is forward and e_2 is backward,
$i_1 < j_2$	if e_1 is backward and e_2 is forward

- For labels, define some total order $<_l$ into triples of labels $(\phi(i_1), \phi(j_1), \phi(i_1, j_1))$

DFS Code (3/3)

- Using "$<_e$" between edges and "$<_l$" between (triples of) labels, the order "$<_t$" between tuples t_1 and t_2 with

  $t_k = (i_k, j_k, \phi(i_k), \phi(j_k), \phi(i_k, j_k))$

  is introduced
- Define $t_1 <_t t_2 \iff$ (i) or (ii)

  (i) $(i_1, j_1) <_e (i_2, j_2)$

  (ii) $(i_1, j_1) = (i_2, j_2)$ and
  $(\phi(i_1), \phi(j_1), \phi(i_1, j_1)) <_l (\phi(i_2), \phi(j_2), \phi(i_2, j_2))$
- The DFS code of a graph is a sequence of tuples ordered by "$<_t$"

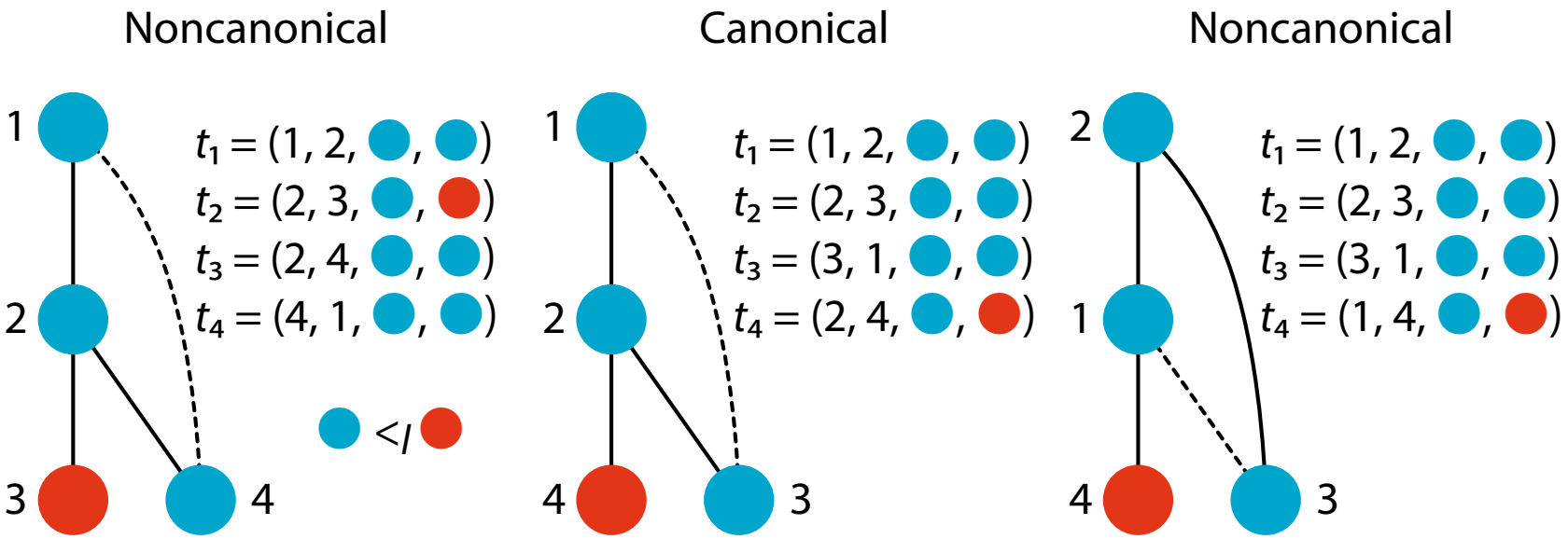
Canonical DFS Code

- Finally, introduce the order "$<$" between two DFS codes $t = (t_1, t_2, \dots, t_m)$ and $t' = (t'_1, t'_2, \dots, t'_n)$
- Define $t < t' \iff$ (i) or (ii)

  (i) $\exists k$ s.t. $0 \leq k \leq \min(m, n)$,
  $t_1 = t'_1,\ t_2 = t'_2,\ \dots,\ t_{k-1} = t'_{k-1},\ t_k < t'_k$

  (ii) $m \leq n$ and $t_1 = t'_1,\ t_2 = t'_2,\ \dots,\ t_m = t'_m$
- The canonical DFS code of a graph G
  $\rightarrow$ its smallest DFS code w.r.t. the order "$<$"

Canonical DFS Code



Rightmost Path Extension

- In the DFS traversal of a graph G, the rightmost path is the path from the root to the “rightmost” leaf with the largest index
- Rightmost path extension achieves systematic candidate graph generation from an existing graph G by either
  (i) adding a backward edge from the rightmost vertex to other vertex on the rightmost path, or
  (ii) adding a forward edge from a vertex on the rightmost path

The gSpan Algorithm

Algorithm 1: gSpan

// $C \leftarrow \emptyset$ for the initial call

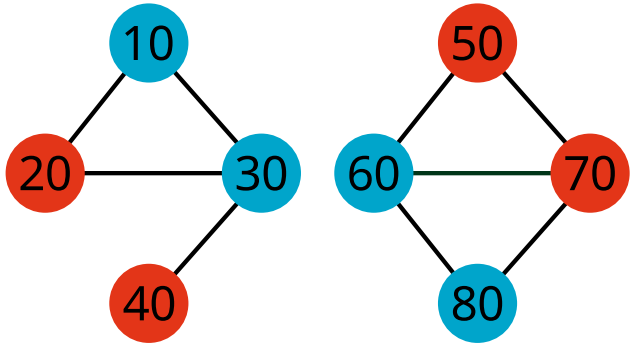
```
1 gSpan(C, D, σ)
2     E ← RightmostPathExtension(C, D)
3     foreach (t, η_t) ∈ E do
4         C ← C ∪ {t}
5         η(C) ← η_t
6         if η(C) > σ and isCanonical(C) then
7             gSpan(C, D, σ)
```

Subprocesses in gSpan

- RIGHTMOSTPATHEXTENSION(C, D)
  - Receive a graph G represented by its DFS code C and a dataset D
  - Return all possible rightmost path extensions of G
    - A set of pairs of tuples and frequencies
      $\mathcal{E} = \{(t_1, \eta_{t_1}), (t_2, \eta_{t_2}), \dots, (t_m, \eta_{t_m})\}$
- ISCANONICAL(C)
  - Receive a DFS code C
  - Return TRUE if C is canonical and FALSE otherwise

Example of Extension

Graph database:



Example of subgraph isomorphisms:

	0	1	2
π_1:	10	30	20
π_2:	60	80	70

Possible extensions:

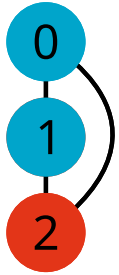
π_1: {⟨2,0,b,a⟩, ⟨1,3,a,b⟩}

π_2: {⟨2,0,b,a⟩, ⟨2,3,b,b⟩, ⟨0,3,a,b⟩}

Current candidate:

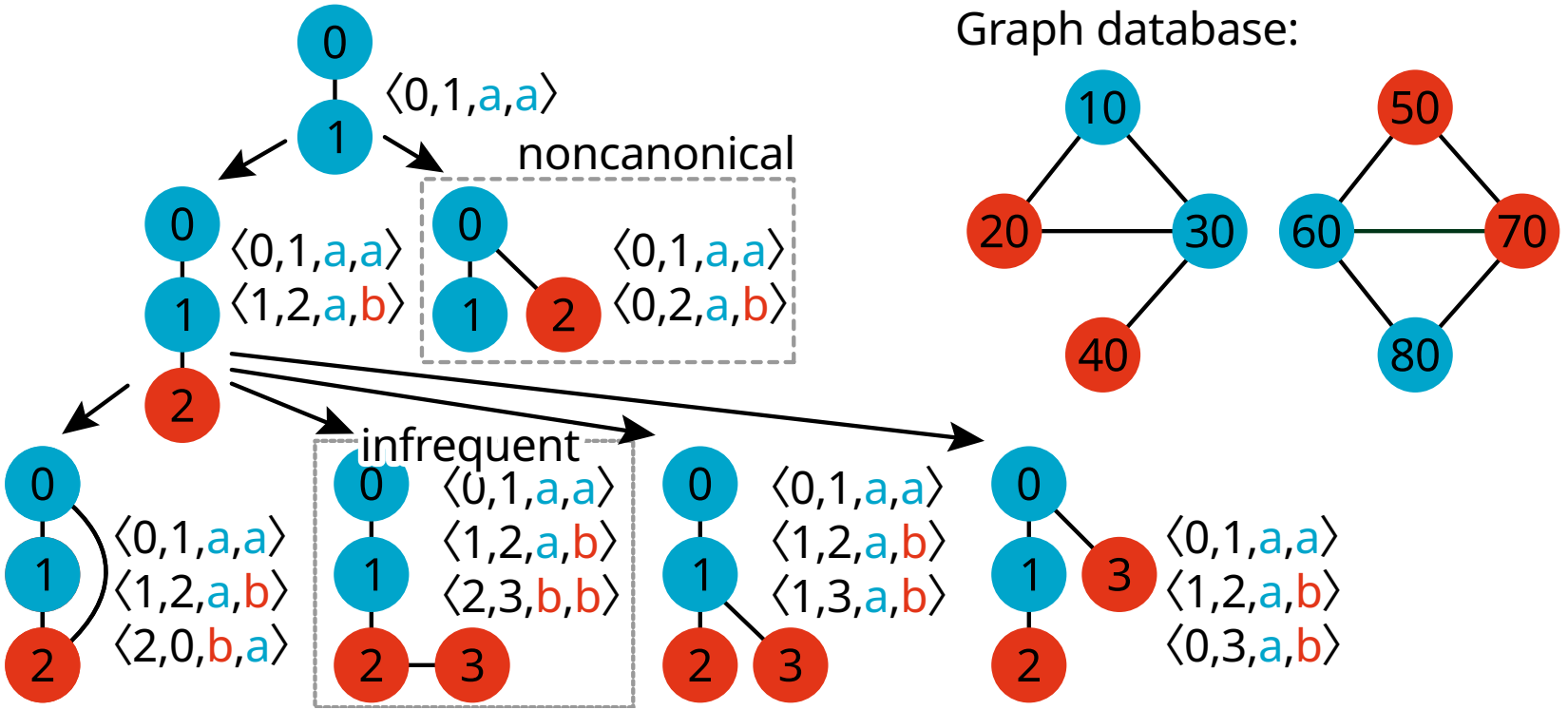
⟨0,1,a,a⟩
⟨1,2,a,b⟩

Extensions:
(⟨2,0,b,a⟩, 2), ...



⟨0,1,a,a⟩
⟨1,2,a,b⟩
⟨2,0,b,a⟩

Example of Enumeration



Conclusion

- gSpan achieves graph mining in an efficient manner
- The keys are:
  - Canonical DFS codes
  - Rightmost path extension
  - Combine them with the Apriori principle

Some Recent Topics in ML

- How to treat graph structures in ML is a challenging task
  - Graph kernels
  - Graph Neural Networks (GNNs), Graph Attention Networks (GATs)
- Recovering graphs from models (feature spaces) is even more challenging
  - Yamada & Sugiyama, Molecular Graph Generation by Decomposition and Reassembling, *ACS Omega* (2023)
- Recently, as LLMs have become powerful, sequence-based graph ML may be effective despite losing topological information
  - Chen et al., Flatten Graphs as Sequences: Transformers are Scalable Graph Generators, *NeurIPS2025*