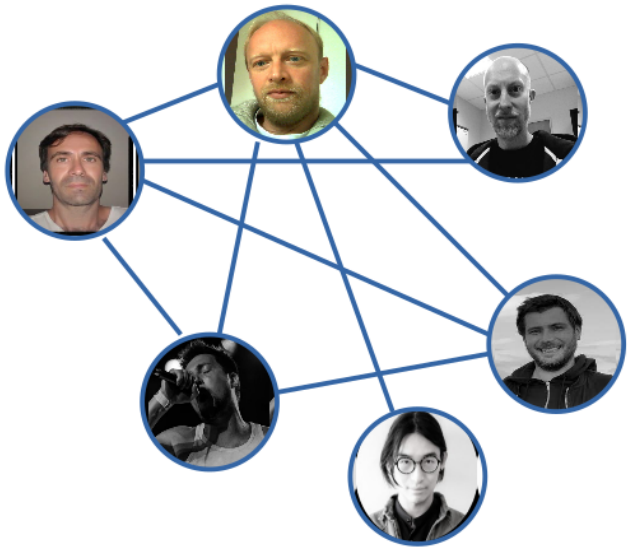
Machine Learning on Graphs
Master 2 Data Science 2025-2026

Stéphane Nicolas / Pierre Héroux / Sébastien Adam / PhD students

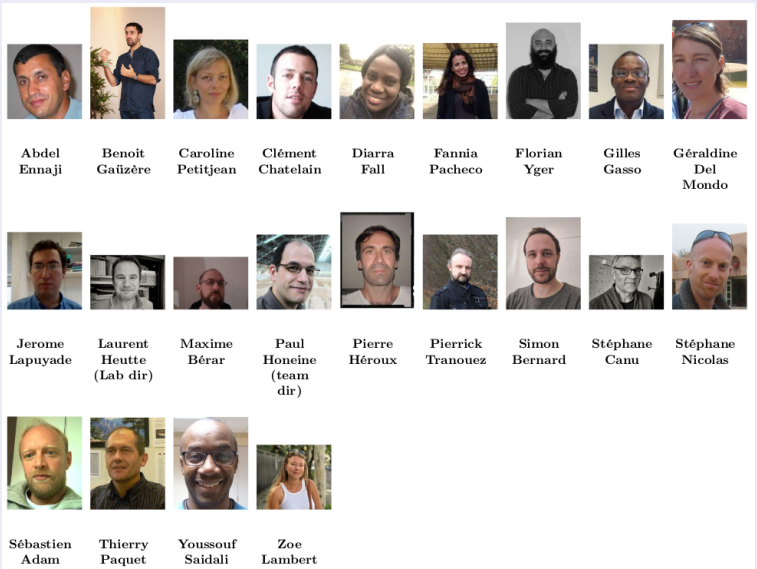


Lecture presentation (1)

Who am I ?

- Sébastien ADAM
  - Full Professor at the University of Rouen Normandy, in the LITIS lab
  - Head of the SD Master (I will sign your diplomas ☺)
  - Deputy Director of the Normastic CNRS federation (LITIS+GREYC)

My research team : the "machine learning" team



Lecture presentation (2)

The "machine learning" team

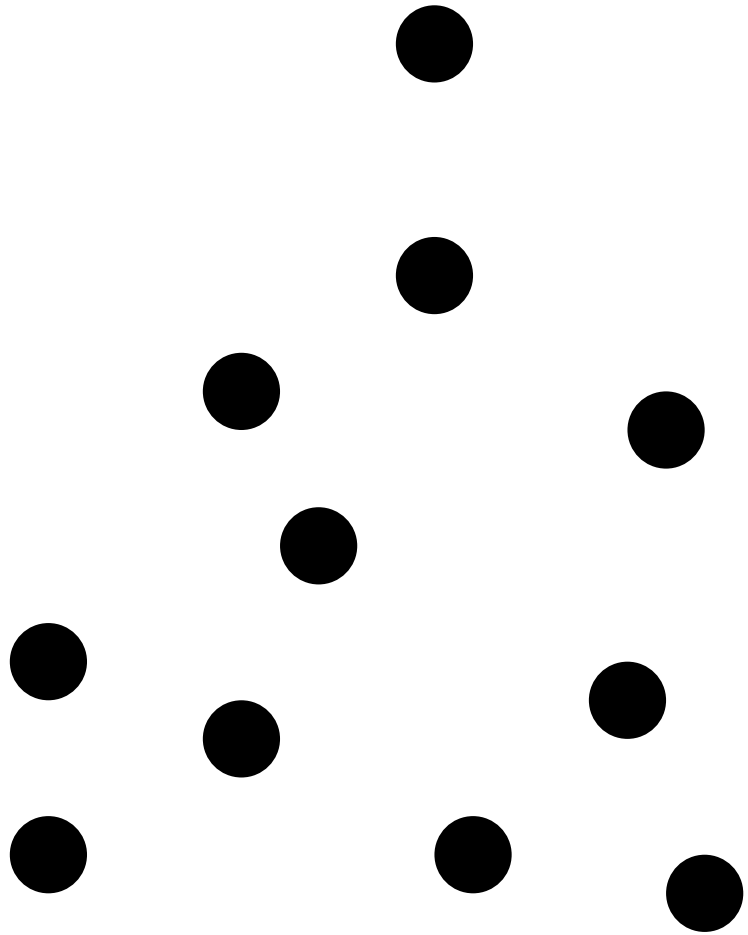
- 22 permanent researchers : 8 PR / 3 MCF HDR / 11 MCF / 1 IR
- ≈ 20 PhD students + ≈ 5 Post-Doc and research engineers
- Target : new ML models for extracting knowledge from raw data :
  - Vectors/Tensors of numbers (signal, image, features...) → statistical learning (optimization, kernels, DL, ensembles, optimal transport...)
  - Sequences of vectors (strings : text, handwritting...) → syntactical learning (HMM, CRF, RNN, LLM, ...)
  - Graphs of vectors (chemistry, social networks, documents...) → **structural (geometric) learning** [Benoit, Clement, Geraldine, Maxime, Paul, Pierre, Stephane, Florian and me]
- Applications : Document image analysis, Medical image analysis, BCI, audio analysis, chemo-informatique, physics...

A graph : what do we talk about ?

- $G = (\mathcal{V}, \mathcal{E}, \mu, \xi)$

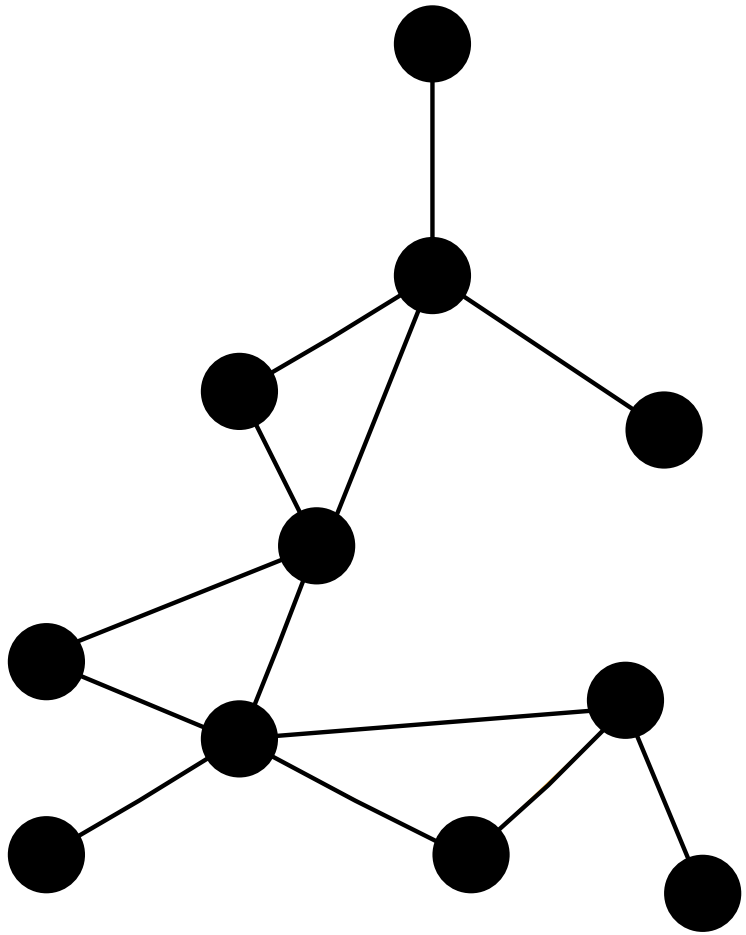
A graph : what do we talk about ?

- $G = (\mathcal{V}, \mathcal{E}, \mu, \xi)$
- $\mathcal{V}$ is the set of nodes



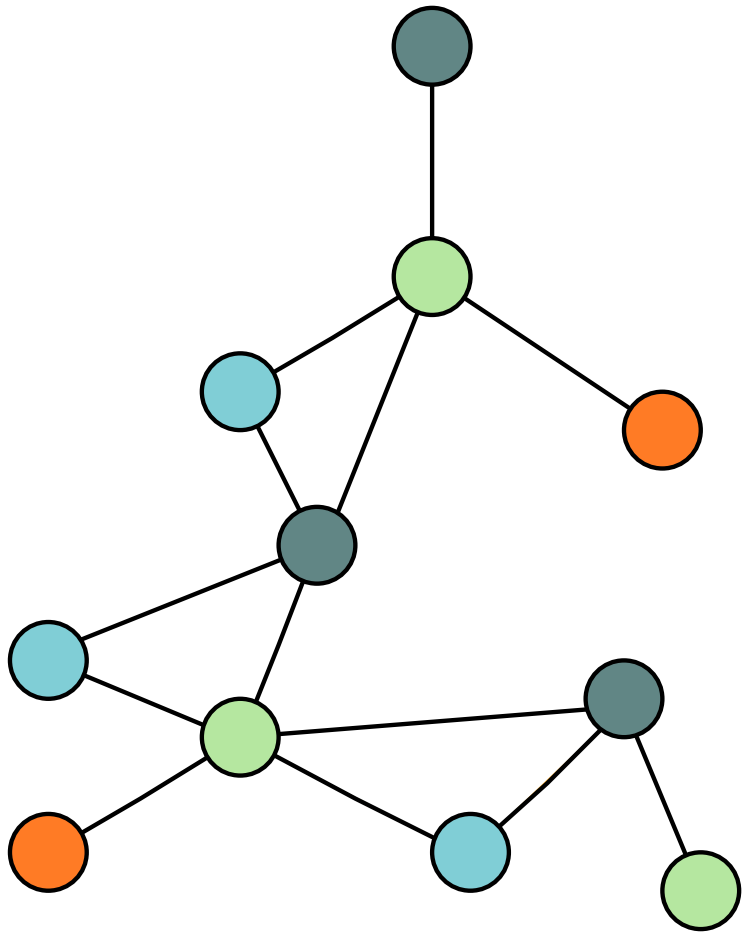
A graph : what do we talk about ?

- $G = (\mathcal{V}, \mathcal{E}, \mu, \xi)$
- $\mathcal{V}$ is the set of nodes
- $\mathcal{E}$ is the set of edges which link node pairs



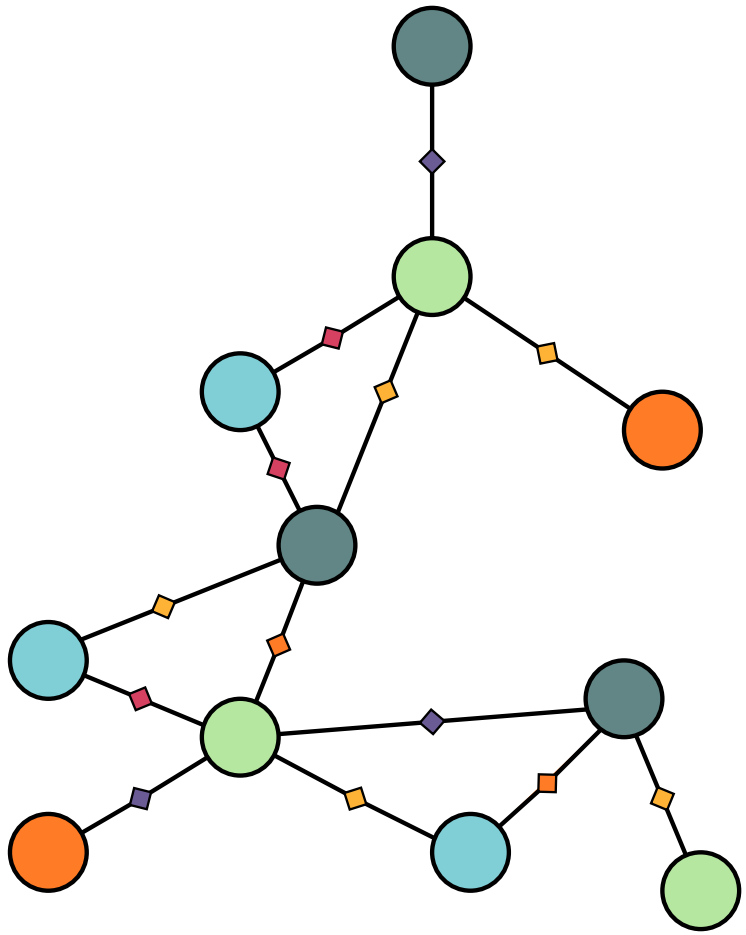
A graph : what do we talk about ?

- $G = (\mathcal{V}, \mathcal{E}, \mu, \xi)$
- $\mathcal{V}$ is the set of nodes
- $\mathcal{E}$ is the set of edges which link node pairs
- Nodes can be attributed by a function μ : the signal on graph



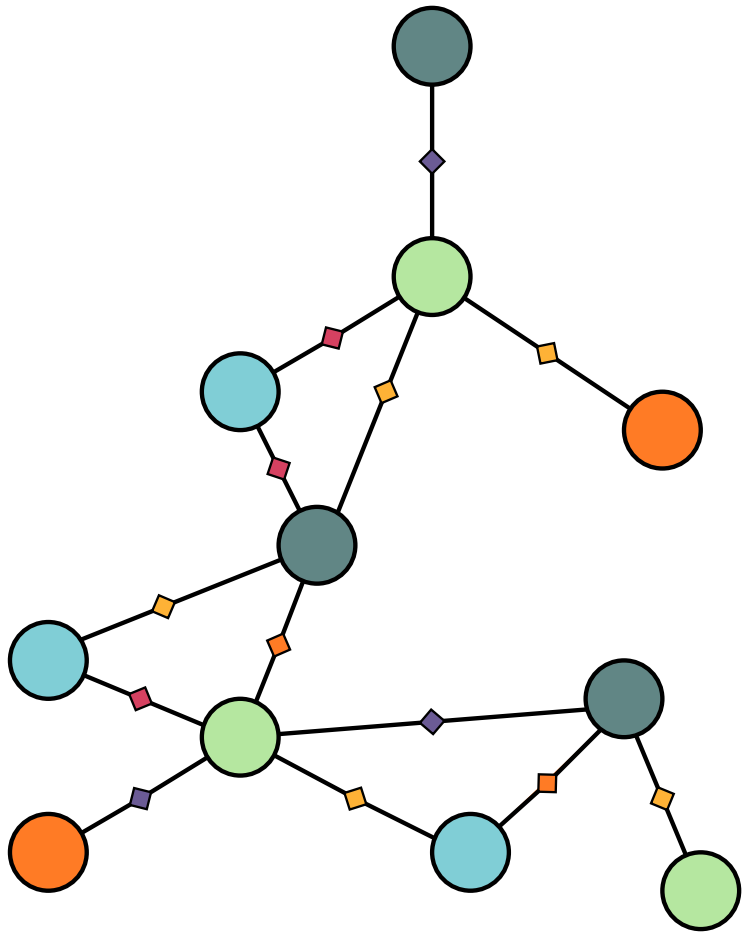
A graph : what do we talk about ?

- $G = (\mathcal{V}, \mathcal{E}, \mu, \xi)$
- $\mathcal{V}$ is the set of nodes
- $\mathcal{E}$ is the set of edges which link node pairs
- Nodes can be attributed by a function μ : the signal on graph
- Edges can be attributed by a function ξ



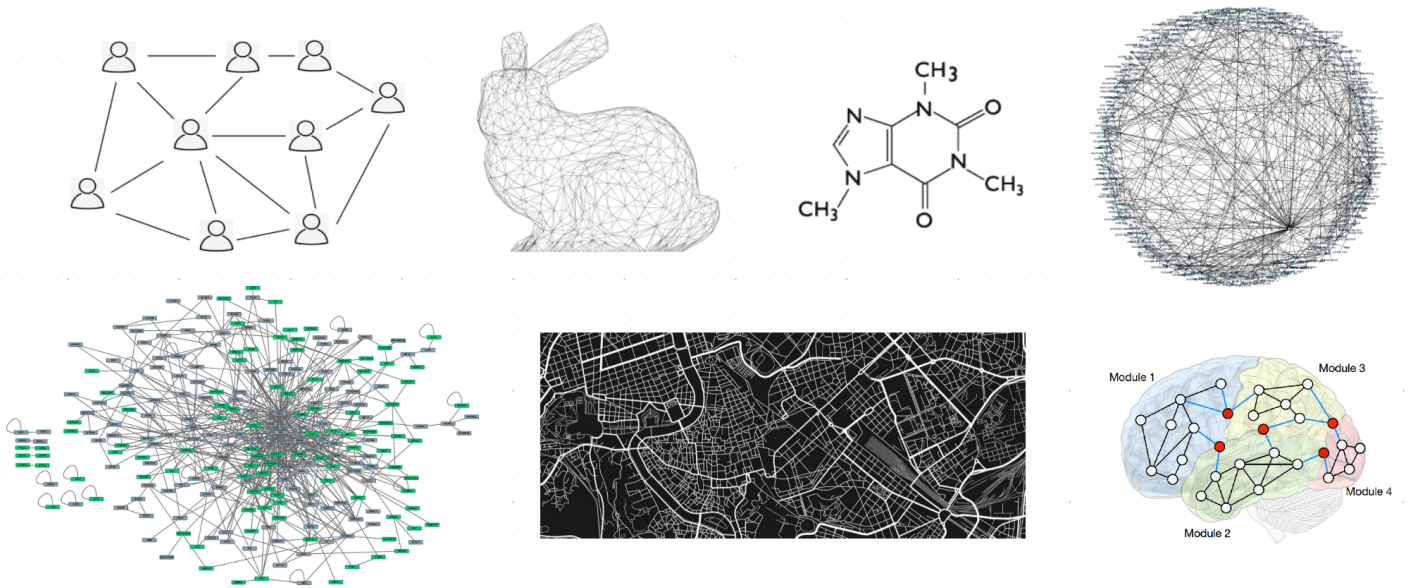
A graph : what do we talk about ?

- $G = (\mathcal{V}, \mathcal{E}, \mu, \xi)$
- $\mathcal{V}$ is the set of nodes
- $\mathcal{E}$ is the set of edges which link node pairs
- Nodes can be attributed by a function μ : the signal on graph
- Edges can be attributed by a function ξ
- NB : graphs can evolve through time



Why talking about graph in a Data Science program ?

- Because it is a very general language for describing and analyzing interacting entities (strong representation power, not limited in dimensions, explainability)

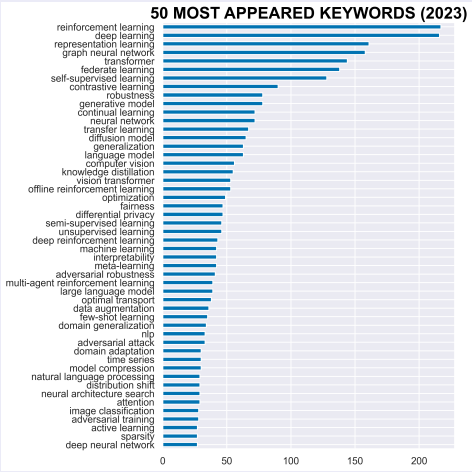


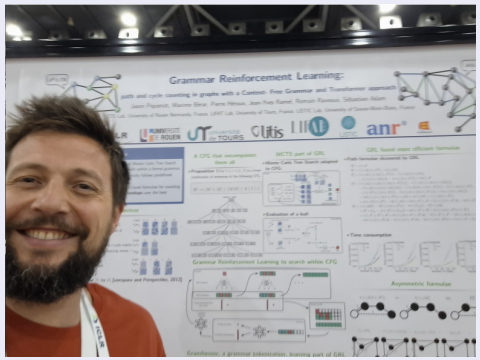
→ Many application domains are concerned by graphs

→ Graphs raise many new machine learning challenges (computational ressources, invariance...)

Context

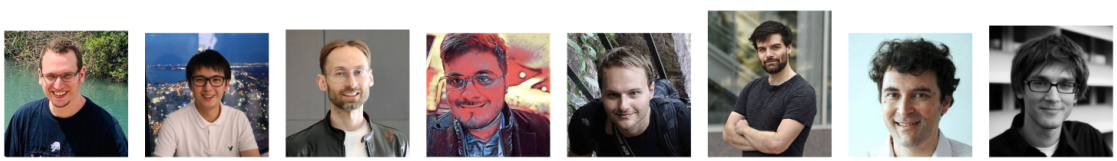
Machine Learning on Graphs : a very hot topic (ICLR data)





What is a poster session in a big ML conference ? `https://photos.app.goo.gl/h4fVEbwp5DiSojHu8`

Machine Learning on Graphs : the Big Chiefs



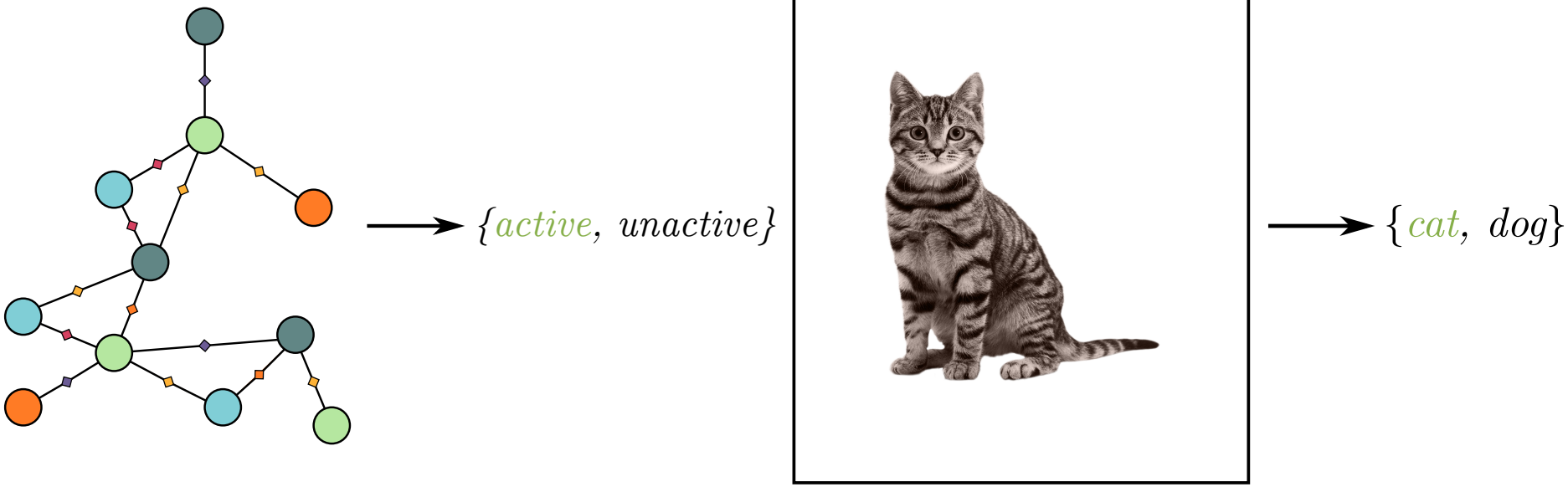
Haggai Maron (nvidia)	Keyulu Xu (MIT)	Michael Bronstein (twitter)	Petar Velickovic (deepmind)	Thomas Kipf (goo-gleBrain)	William Hamilton (McGill)	Xavier Bresson (U-Singapore)	jure-leskovec (stanford)

The best presentation I ever seen : `https://www.youtube.com/watch?v=w6Pw4MOzMuo`

What kind of predictions can we make on graphs ?

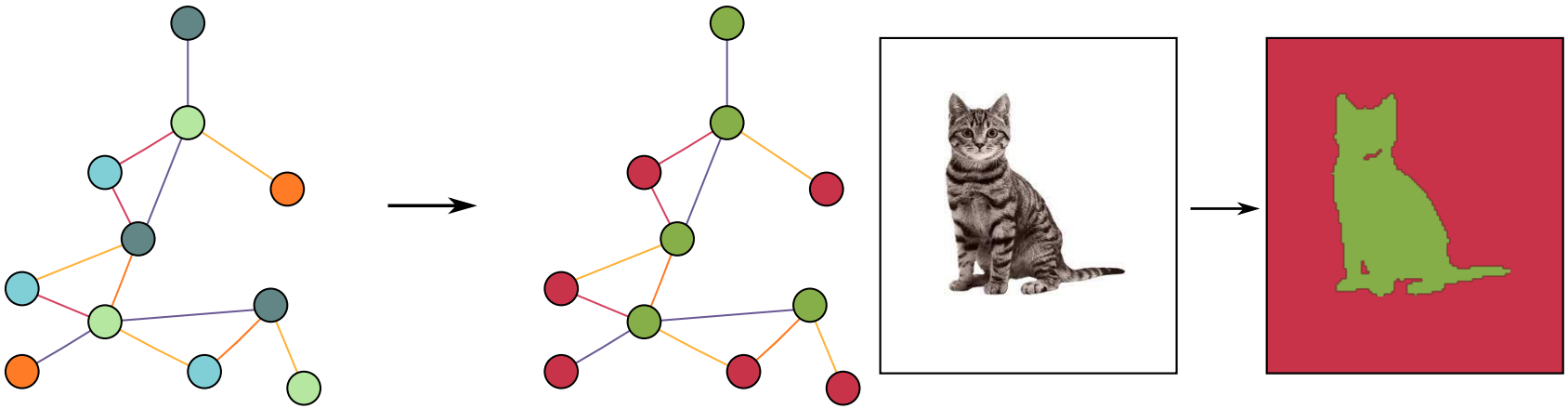
What kind of predictions can we make on graphs ?

- Classification/regression of graphs (in : a graph / out : a label)



What kind of predictions can we make on graphs ?

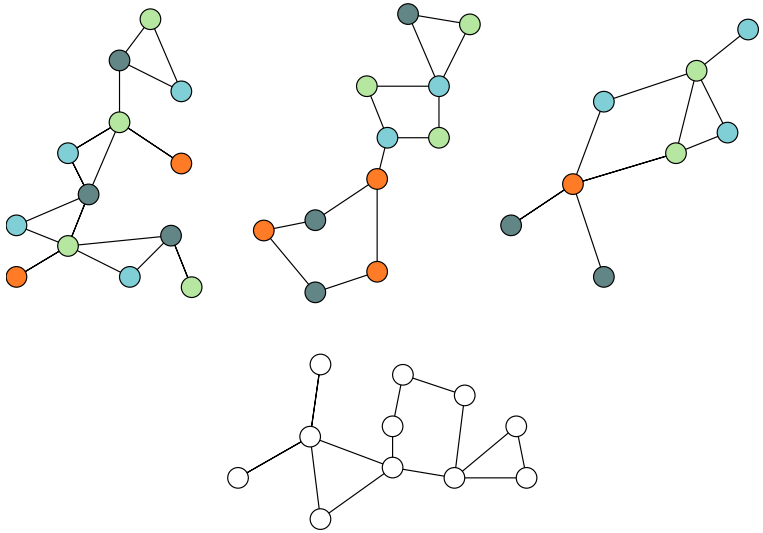
- Classification/regression of graphs (in : a graph / out : a label)
- Classification/regression of nodes (in : a graph / out : nodes labels

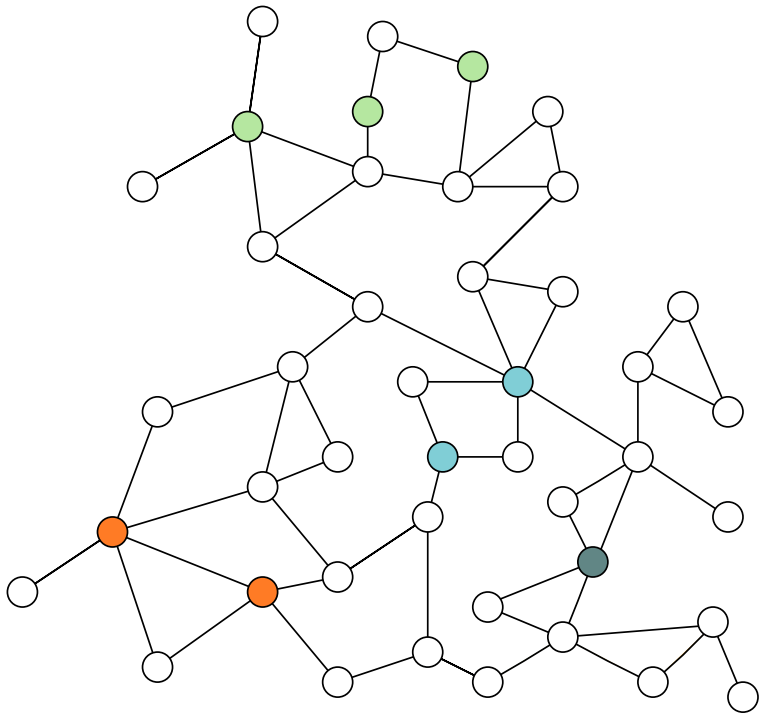


What kind of predictions can we make on graphs ?

Two cases

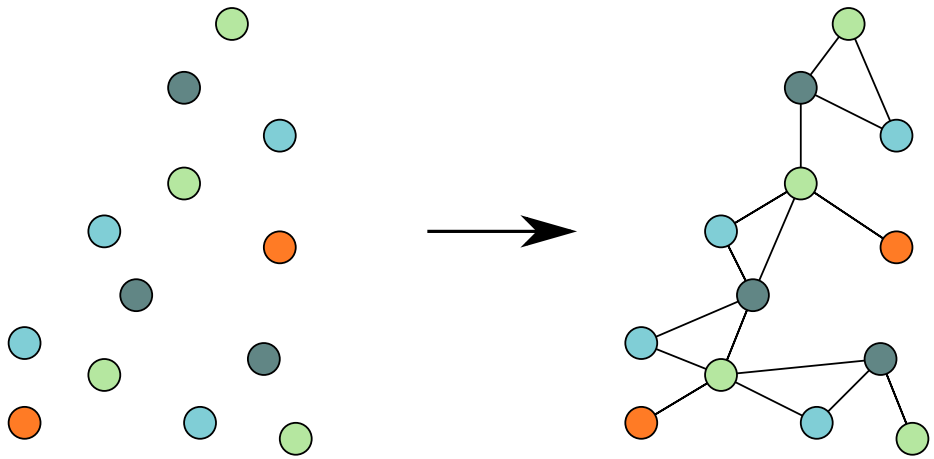
- Inductive tasks : decision on a fully unknown graph
- Transductive tasks : decision on a partially known graph (semi-supervised)





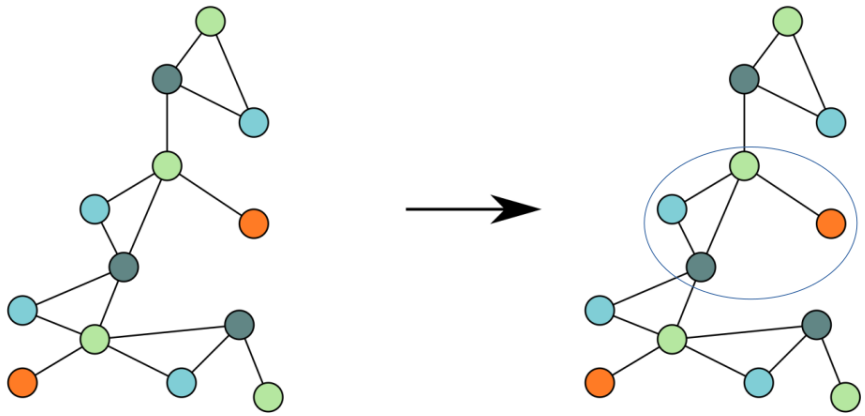
What kind of predictions can we make on graphs ?

- Classification/regression of graphs (in : a graph / out : a label)
- Classification/regression of nodes (in : a graph / out : nodes labels
- link prediction (in : a pair of nodes / out : presence/absence of a link)



What kind of predictions can we make on graphs ?

- Classification/regression of graphs (in : a graph / out : a label)
- Classification/regression of nodes (in : a graph / out : nodes labels
- link prediction (in : a pair of nodes / out : presence/absence of a link)
- Community detection (in : a graph / out : a subgraph) : ≈ clustering

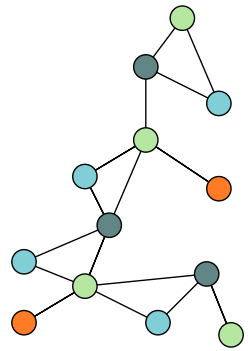


What kind of predictions can we make on graphs ?

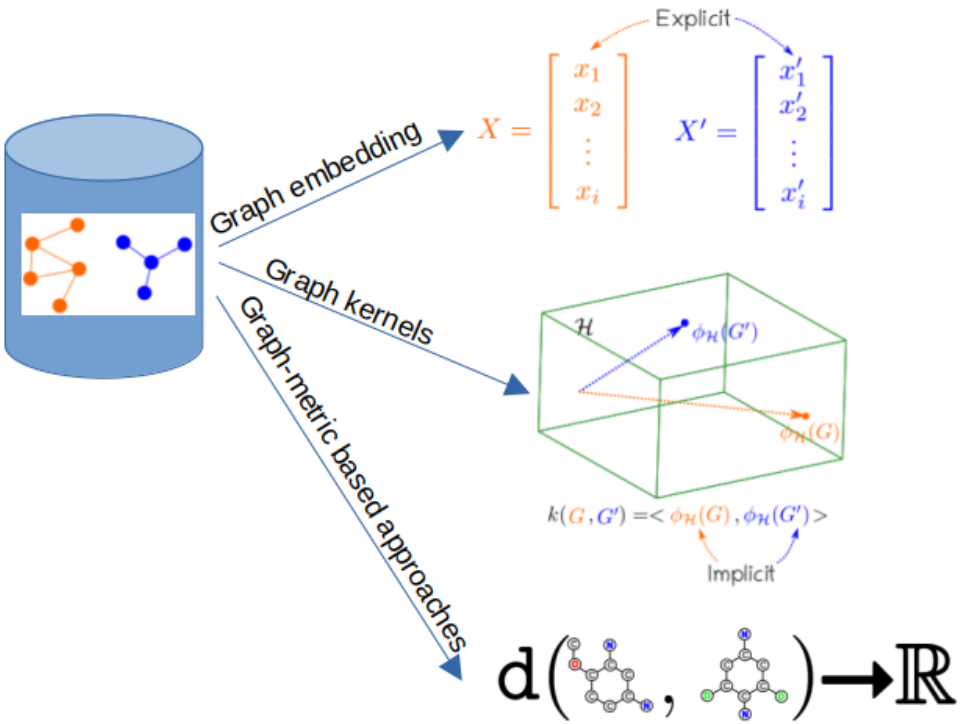
- Classification/regression of graphs (in : a graph / out : a label)
- Classification/regression of nodes (in : a graph / out : nodes labels
- link prediction (in : a pair of nodes / out : presence/absence of a link)
- Community detection (in : a graph / out : a subgraph) : ≈ clustering
- Graph generation (in : image or text / out : a graph : ≈ graphGPT)

A stop sign is flying in blue skies.

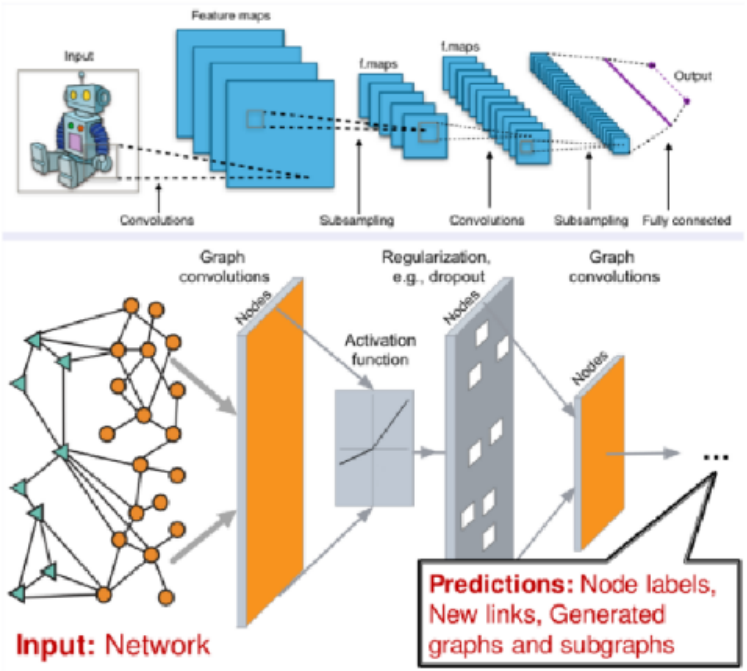
John is in the kitchen...
...The cat is near John



What kind of approaches to solve graph ML tasks ?



Before the Deep learning Era



Recent approaches

Outline of the lecture

Objectives of the lecture : metrics on graphs (1)

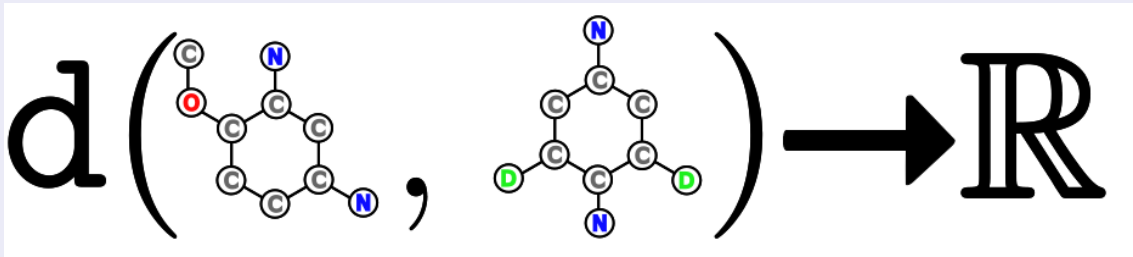
Computing graph (dis)similarity for PR applications

Computing similarity or dissimilarity between objects : basic requirement for machine learning (linear SVM, attention-based models)

- In $\mathbb{R}^n$: easy : Minkowski distance

$$\|x-y\|_p = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}} : (p = 1 : \text{Manhattan},\ p = 2 : \text{Euclidian},\ p = \infty : \text{Tchebychev - max})$$

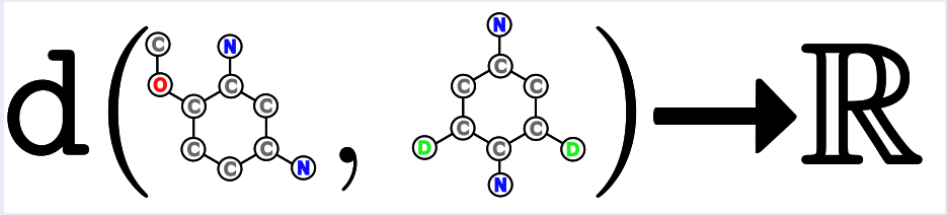
- In the world of sequences : Levenshtein, Dynamic Time Warping
- In the world of graphs : Objective of the lecture



- Any idea ?

Objectives of the lecture : metrics on graphs (2)

Existing Approaches



Many contributions in the literature, with 5 families of approaches :

- Explicit graph embedding methods (graph features) → Pierre's lecture
- Implicit graph embedding methods : kernels → Pierre's lecture
- Spectral methods : → In two lectures
- Matching based methods : correspondences between vertices and edges of both graphs are used to quantify "common" parts of graphs or to determine operations to transform one graph into the other
  - Exact approaches (strict correspondence)
  - Inexact (error-tolerant) approaches : can compare non-identical graphs

  Gives a dissimilarity + a mapping → a step toward "explainable" AI
- Very recently : (deep) learning-based methods (Aldo's lecture)

Outline of the lecture

Definitions and notations

Attributed graphs

A labeled graph G is a 4-tuple $G = (V, E, \mu, \zeta)$, where :

- V is the set of nodes,
- $E \subseteq V \times V$ is the set of edges
- $\mu : V \to L_V$ is a function assigning labels to the nodes, and
- $\zeta : E \to L_E$ is a function assigning labels to the edges.

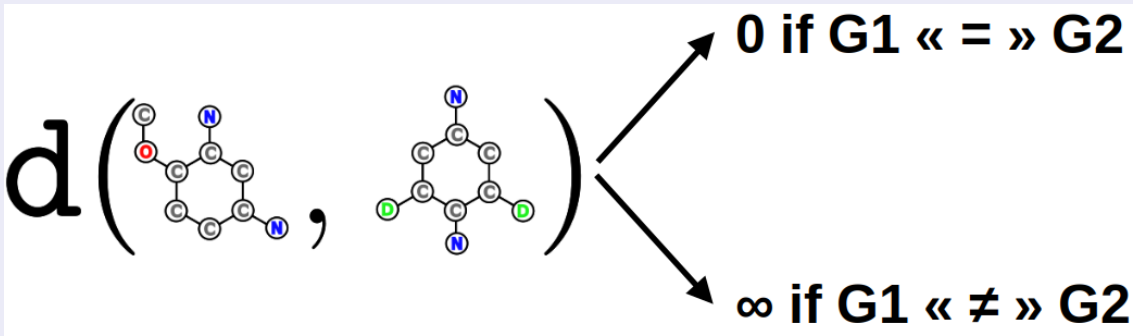
L_V and L_E denote the set of node and edge labels which can be :

- a finite set of symbolic labels L = {x,y, z, . . .}
- a vector space L = $\mathbb{R}^n$
- and everything you can imagine

This definition allows to handle arbitrarily structured graphs.

The "simplest" (dis)similarity

0/∞ distance



- Requires to test "equality" of graphs
- Well known problem already met with S. Nicolas : graph isomorphism

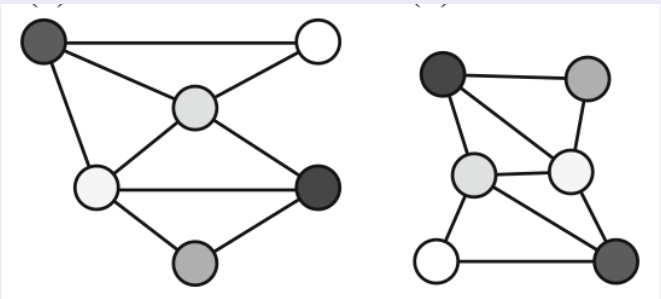
Graph isomorphism

Definition (in the case of attributed graphs)

Let $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$ be two graphs. A graph isomorphism is a bijective mapping $f : V_1 \to V_2$ satisfying :

1. $\mu_1(u) = \mu_2(f(u)) \ \forall u \in V_1$
2. $e_1 = (u, v) \in E_1 \Rightarrow e_2 = (f(u), f(v)) \in E_2$ with $\zeta(e_1) = \zeta_2(e_2) \ \forall e_1 \in E_1$
3. $e_2 = (u, v) \in E_2 \Rightarrow e_1 = (f^{-1}(u), f^{-1}(v)) \in E_1$ with $\zeta_1(e_1) = \zeta_2(e_2) \ \forall e_2 \in E_2$

G_1 and G_2 are isomorphic is an isomorphism f exists between V_1 and V_2



- Not known to be solvable in polynomial time nor to be NP-complete
- State of the art algorithm : VF2-3 (Foggia) or LAD (Solnon)
- Testing if an isomorphism exists leads to a binary dissimilarity measure

Graph isomorphism : a quick aside

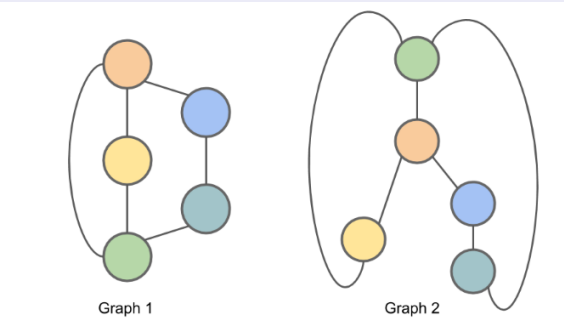
The Weisfeiler-Lehman (WL) test

- The WL test is a heuristic algorithm for testing graph isomorphism
- The "classical" Weisfeiler-Lehman test : color refinement
- Each node has a state (color) that is refined using the neighbors state
- Output : a graph embedding $WL(G)$ corresponding to the state of every node
- For 2 graphs G_1 and G_2
  - if $WL(G_1) \neq WL(G_2)$, they are not isomorphic.
  - else, we don't really know...

Algorithm *1-WL (color refinement)*
Input: $G = (V, E, X_V)$
1. $c_v^0 \leftarrow \text{hash}(X_v)$ for all $v \in V$
2. **repeat**
3. $\quad c_v^\ell \leftarrow \text{hash}(c_v^{\ell-1}, \{\!\{c_w^{\ell-1} : w \in \mathcal{N}_G(v)\}\!\})\ \forall v \in V$
4. **until** $(c_v^\ell)_{v \in V} = (c_v^{\ell-1})_{v \in V}$
5. **return** $\{\!\{c_v^\ell : v \in V\}\!\}$

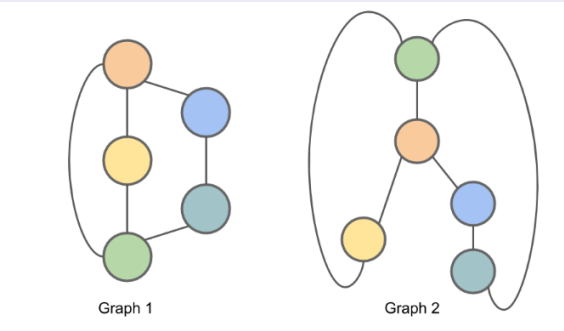
Graph isomorphism

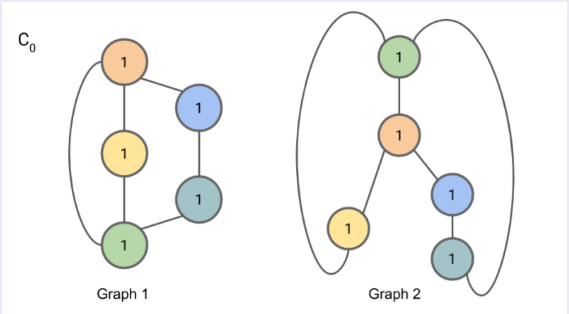
Example of the steps for 2 isomorphic graphs



Graph isomorphism

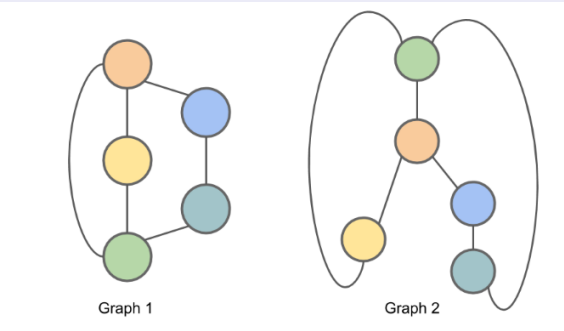
Example of the steps for 2 isomorphic graphs

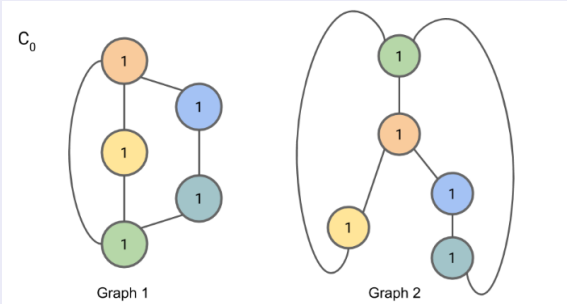


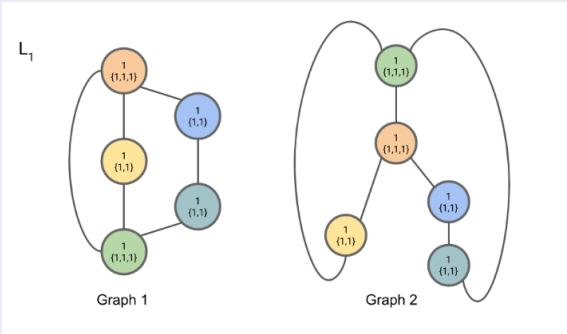


Graph isomorphism

Example of the steps for 2 isomorphic graphs

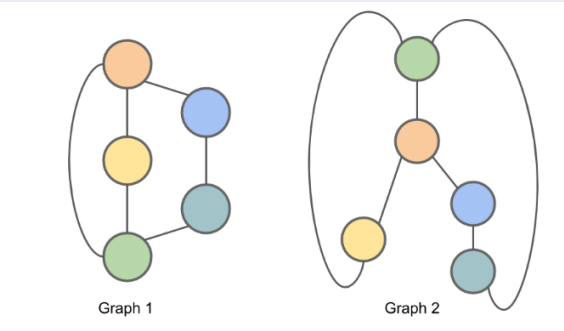


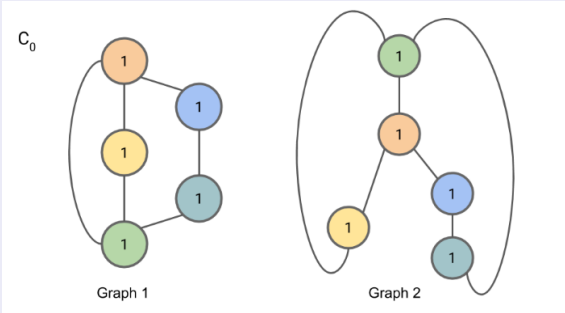


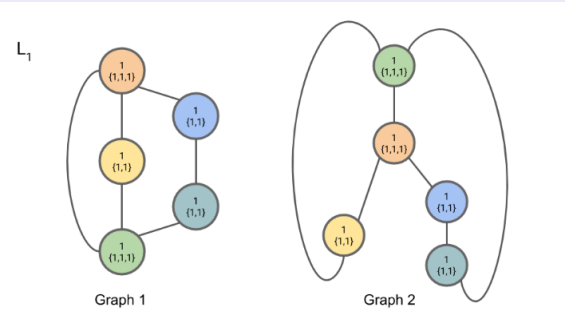


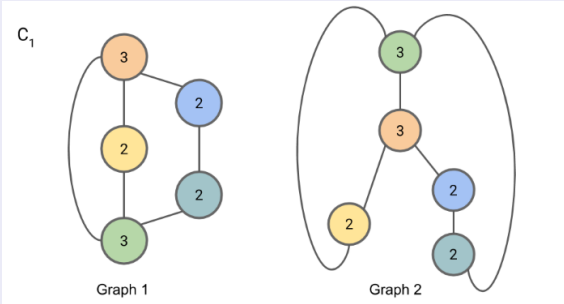
Graph isomorphism

Example of the steps for 2 isomorphic graphs



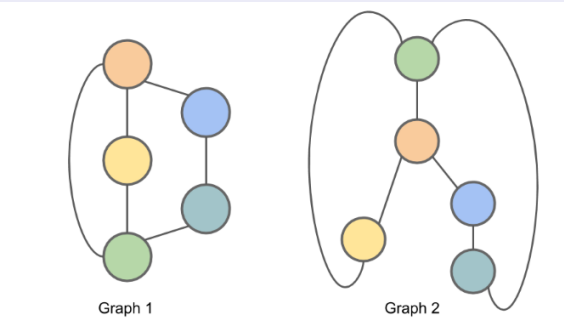


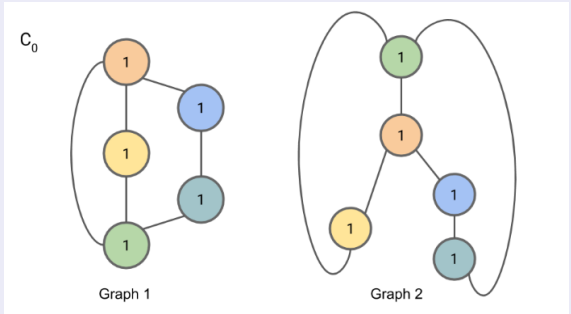


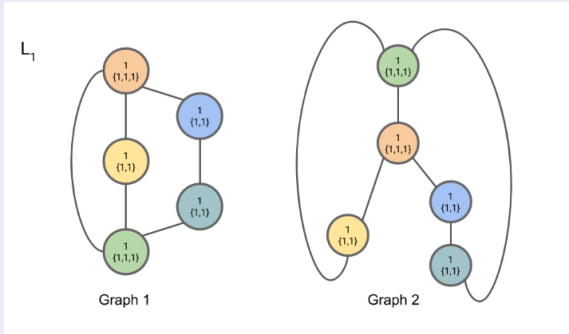


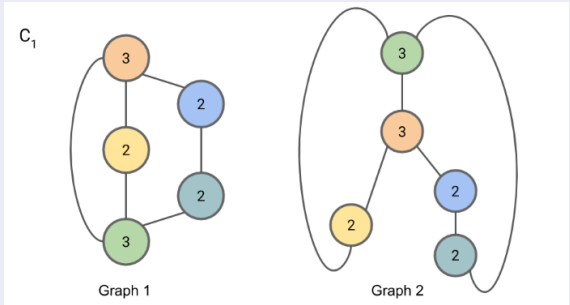
Graph isomorphism

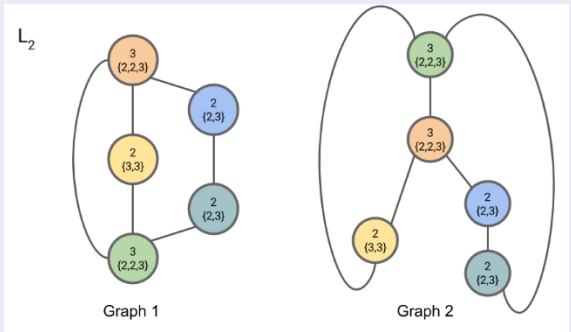
Example of the steps for 2 isomorphic graphs





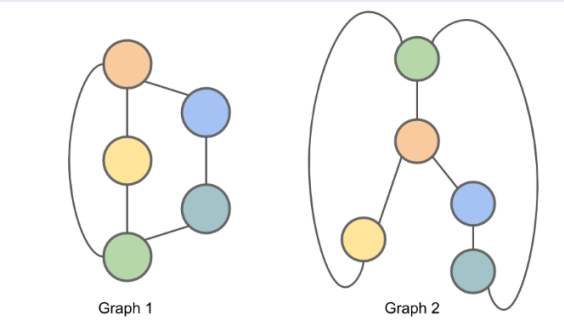


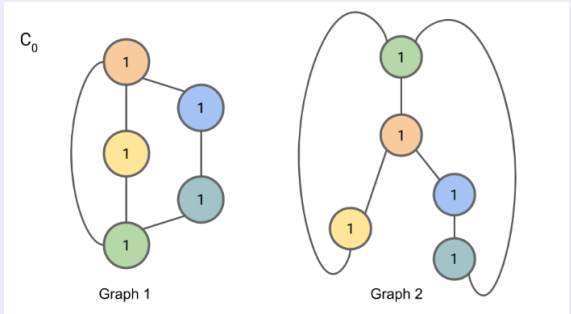


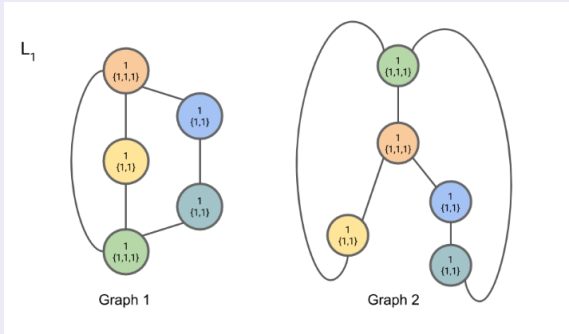


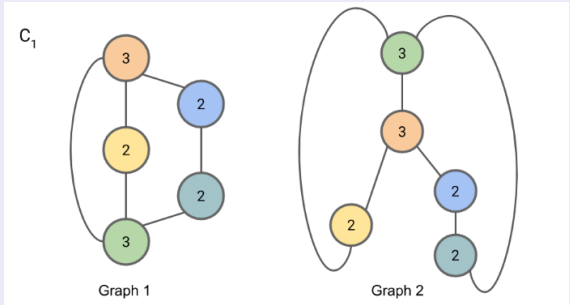
Graph isomorphism

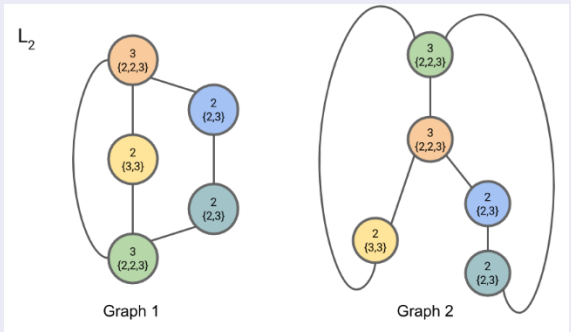
Example of the steps for 2 isomorphic graphs

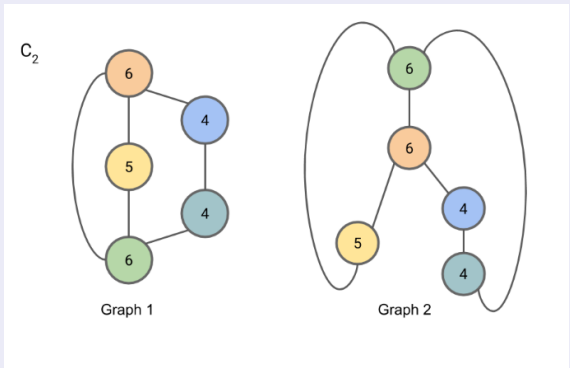






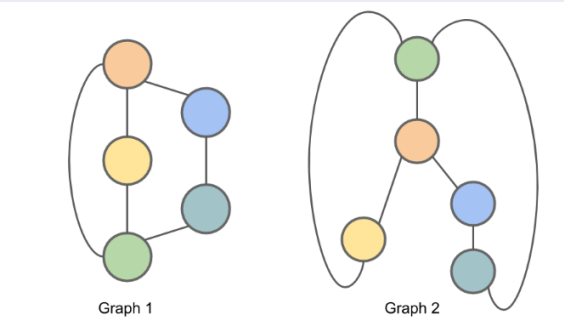


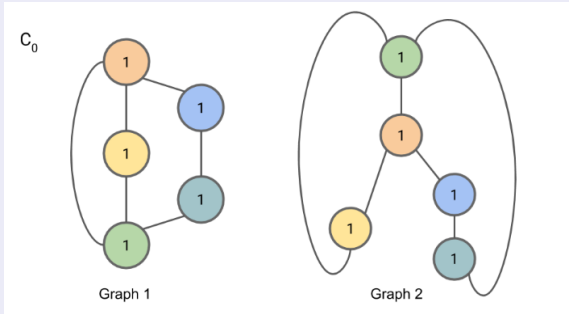


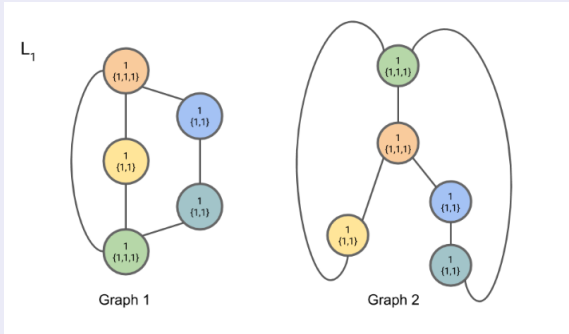


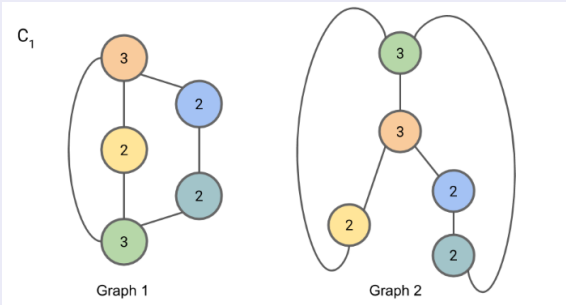
Graph isomorphism

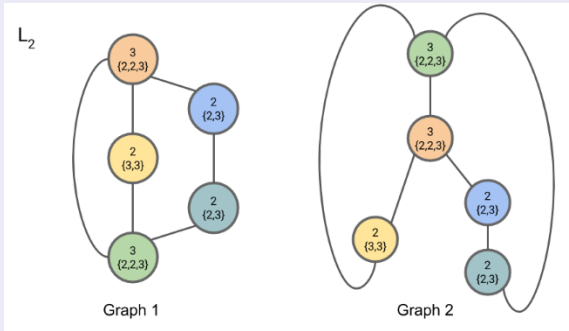
Example of the steps for 2 isomorphic graphs

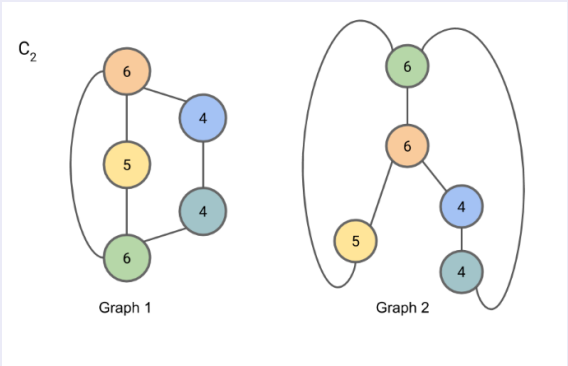


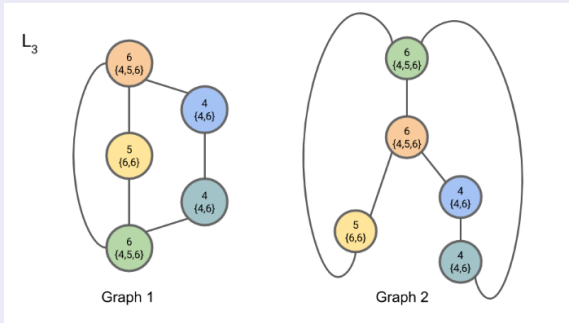






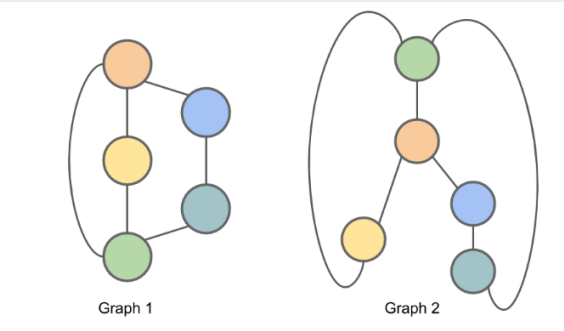


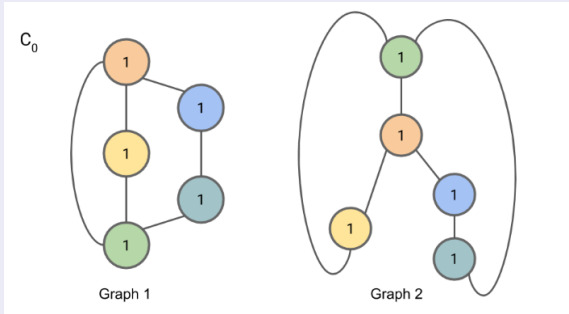


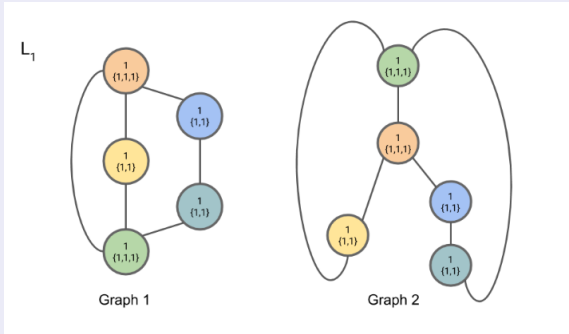


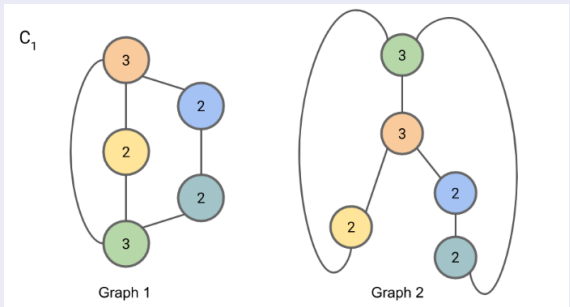
Graph isomorphism

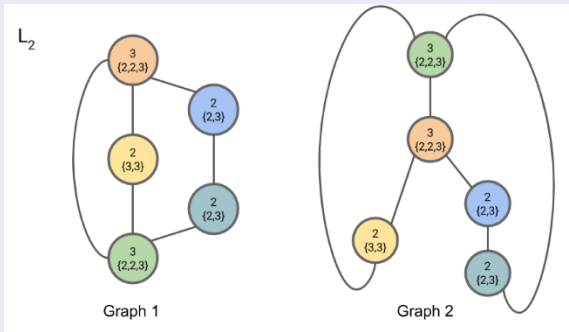
Example of the steps for 2 isomorphic graphs

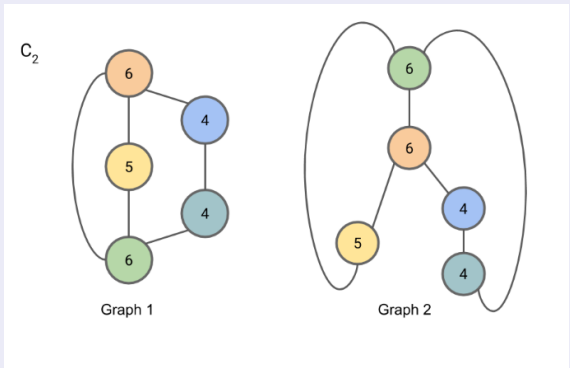


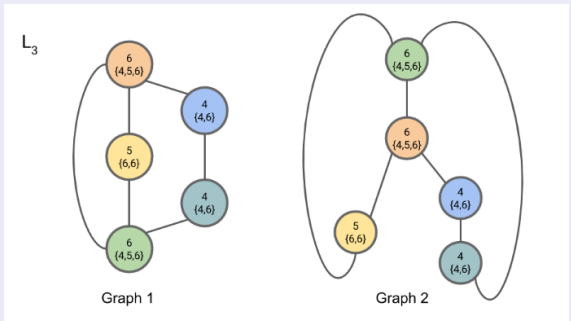


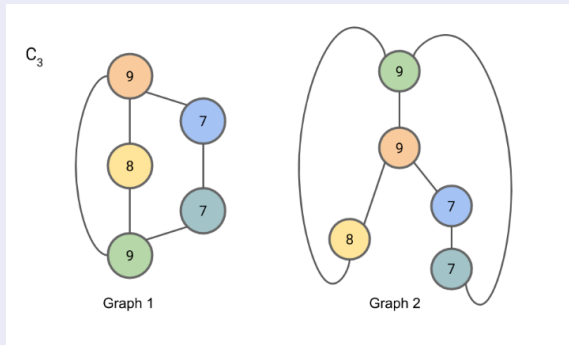






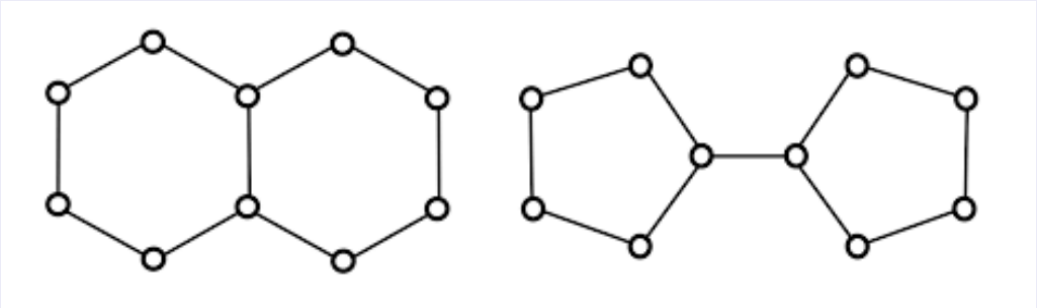






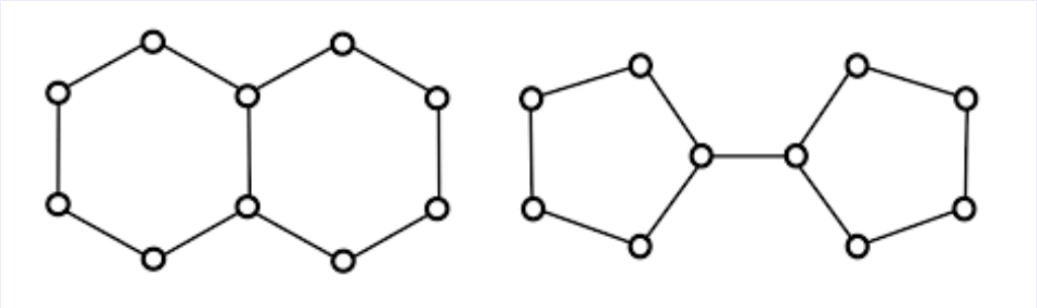
Graph isomorphism

Another example for training yourself on the Decalin/Bicyclopentyl



Graph isomorphism

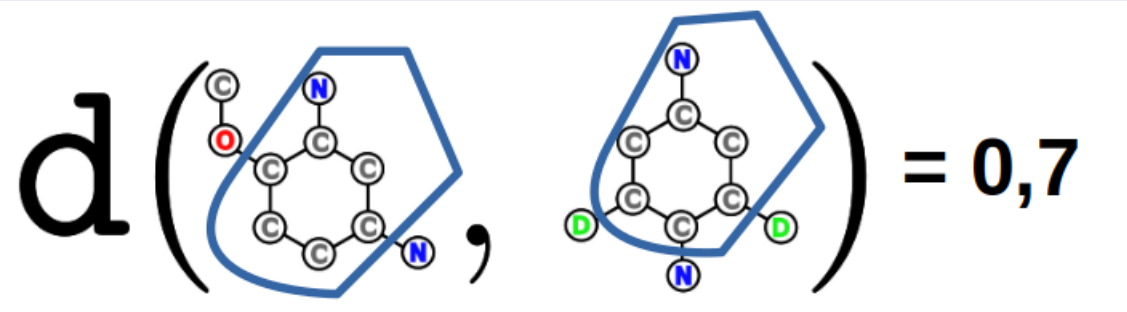
Another example for training yourself on the Decalin/Bicyclopentyl



- In a few weeks, we will see that many SOTA GNN (typically Message Passing Neural Network) are limited to 1-WL expressive power : they are unable to distinguish these 2 graphs !
- NB : some very recent GNN's are based on k-order WL test which apply the same procedure but on k-tupes of nodes (Jason's lecture)
- PB : testing isomorphism leads to a $\{0, 1\}$ dissimilarity measure
- Other idea : characterize common sub-structures (subgraphs)

A "richer" (dis)similarity

Using common sub-structures



- Requires to find common substructures
- Needs to test subgraph isomorphism

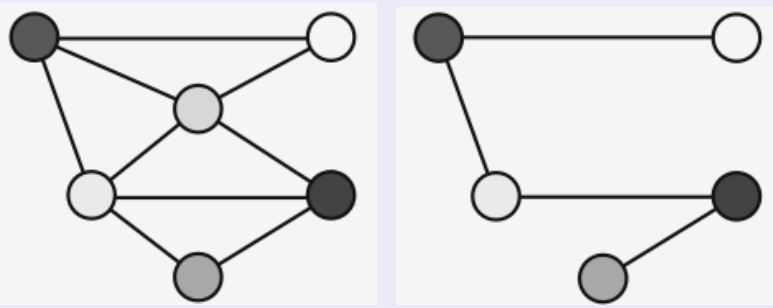
Definitions and notations

Subgraphs

Let $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$ be graphs. G_1 is a subgraph of G_2, denoted by $G1 \subseteq G2$, if :

1. $V_1 \subseteq V_2$
2. $E_1 \subseteq E_2$
3. $\mu_1(u) = \mu_2(u)$ for all $u \in V_1$, and
4. $\zeta_1(e) = \zeta_2(e)$ for all $e \in E_1$.

By replacing the second condition by a more stringent one :
(2') $E_1 = E2 \cap V_1 \times V_1$
G_1 becomes an induced subgraph of G_2



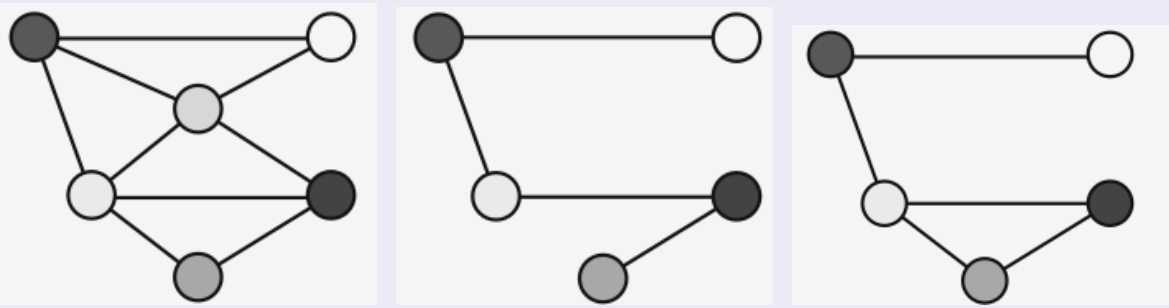
Definitions and notations

Subgraphs

Let $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$ be graphs. G_1 is a subgraph of G_2, denoted by $G1 \subseteq G2$, if :

1. $V_1 \subseteq V_2$
2. $E_1 \subseteq E_2$
3. $\mu_1(u) = \mu_2(u)$ for all $u \in V_1$, and
4. $\zeta_1(e) = \zeta_2(e)$ for all $e \in E_1$.

By replacing the second condition by a more stringent one :
(2') $E_1 = E2 \cap V_1 \times V_1$
G_1 becomes an induced subgraph of G_2

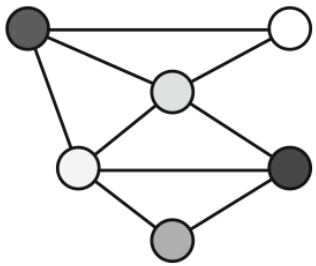


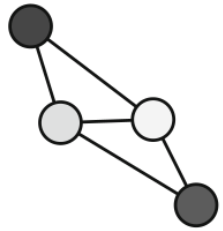
Maximum common subgraph - mcs (1)

Definition of subgraph isomorphism

Let $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$ be two graphs. A subgraph isomorphism is an injective function $f : V_1 \to V_2$ from G_1 to G_2 if there exists a subgraph $G \subseteq G_2$ such that f is a graph isomorphism between G_1 and G

Remark : the definition can be extended to induced subgraphs





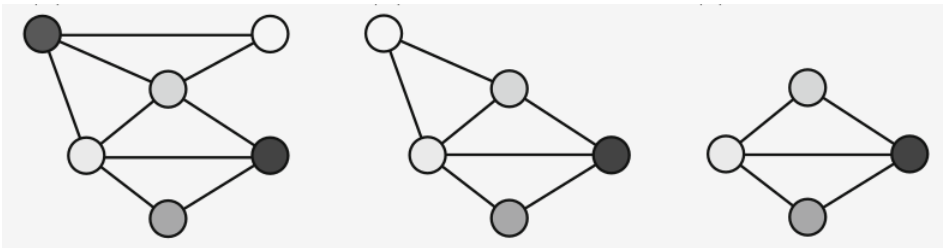
Remarks

- Subgraph isomorphism is a harder problem than graph isomorphism which "only" look for permutations
- Remark : contrary to graph isomorphism, subgraph isomorphism is proven to be NP-complete

Maximum common subgraph - mcs (2)

Definition of maximum common subgraph

Let $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$ be two graphs. A common subgraph of G_1 and G_2 is a graph $G = (V, E, \mu, \zeta)$ such that there exists subgraph isomorphism from G to G_1 and from G to G_2. G is a maximum common subgraph of G_1 and G_2 ($mcs(G_1, G_2)$) if there exists no other common subgraph of G_1 and G_2 that has more nodes than G.



Remarks

- There may be more than one mcs
- mcs = maximal part of graphs identical in structure and labels
- A lot of methods exist for computing *mcs* : let's see a old but standard one

Maximum common subgraph - mcs (3)

Step one : build an association Graph (another graph)

- Nodes for"compatible" pair of nodes v_{1i} in G_1 and v_{2j} in G_2
- Edge are created between (v_{1i}, v_{2j}) and (v_{1k}, v_{2l}) if an isomorphism exists between (v_{1i}, v_{1k}) and (v_{2j}, v_{2l})

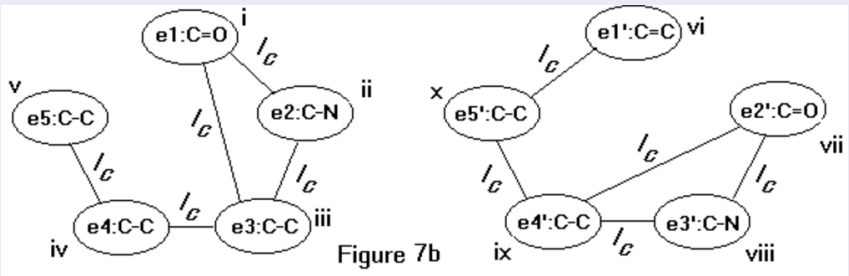
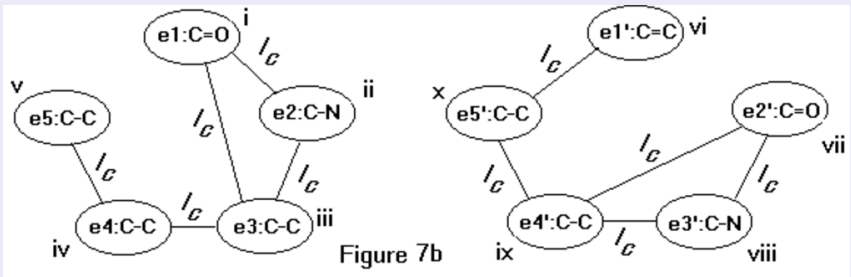


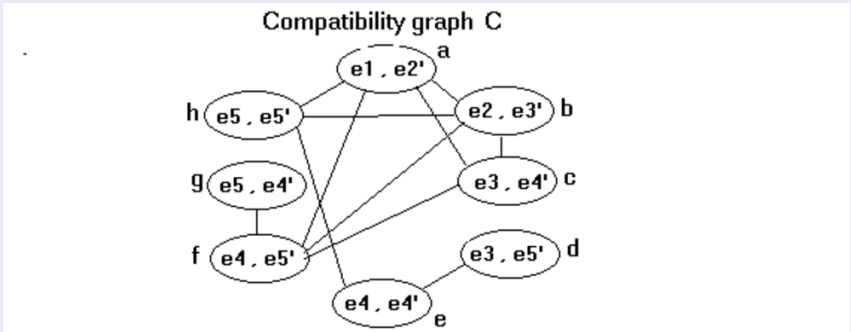
Figure 7b

Maximum common subgraph - mcs (3)

Step one : build an association Graph (another graph)

- Nodes for"compatible" pair of nodes v_{1i} in G_1 and v_{2j} in G_2
- Edge are created between (v_{1i}, v_{2j}) and (v_{1k}, v_{2l}) if an isomorphism exists between (v_{1i}, v_{1k}) and (v_{2j}, v_{2l})



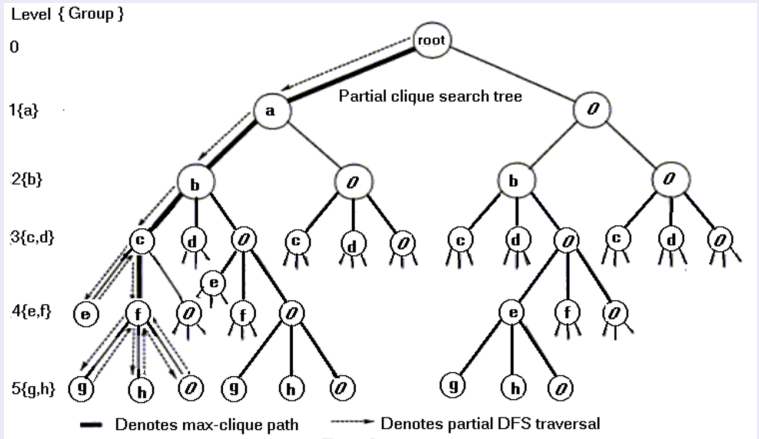


- In this graph, the MCS is the largest ...

Maximum common subgraph - mcs (4)

Tree search in the association graph (Durand-Pasari)

- Groups are computed according to similar v_{1i} in the pairs
- A search tree is explored in a deep first fashion to identify the longest path (i.e. the maximum clique)



- Many refinements are possible using LB and UB (from domain heuristic, node sorting...) to reduce the exploration of the tree
- Example : a LB can be obtained using a clique search in a reduced association graph (including chemical constraints)

Maximum common subgraph - mcs (5)

Dissimilarity computed from mcs

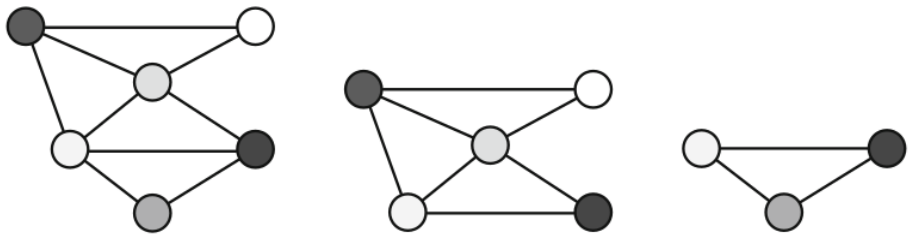
Various dissimilarity measures can be computed from mcs properties :

- $d_{mcs}(G_1, G_2) = 1 - \frac{|mcs(G_1,G_2)|}{max(|G_1|,|G_2|)}$ which is a metric, in $[0,1]$
- $d_{WGU}(G_1, G_2) = 1 - \frac{|mcs(G_1,G_2)|}{|G_1|+|G_2|-|mcs(G_1,G_2)|}$ which is a metric, in $[0,1]$
  - The denominator is the size of the union in the set-theoretic sense
  - Size changes in the smallest graph are taken into account
  - Let's imagine
    - $|mcs(G_1, G_2)| = 10$,
    - $|G_1| = 100$
    - $|G_2|$ varies from 100 to 10
- $d_{UGU}(G_1, G_2) = |G_1| + |G_2| - 2|mcs(G_1, G_2)|$ which is not normalized in $[0,1]$

Minimum Common Supergraph - MCS (1)

Definition of MCS

Let $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$ be graphs. A common supergraph of G_1 and G_2 is a graph $G = (V, E, \mu, \zeta)$ such that there exists sugraph isomorphisms from G_1 to G and from G_2 to G. G is the minimum common supergraph of G_1 and G_2 if there exists no other common supergraph with less nodes than G.



Remarks

- MCS and mcs relation : $|MCS(G_1, G_2)| = |G_1| + |G_2| - |mcs(G_1, G_2)|$
- $d_{UGU}(G_1, G_2) = |MCS(G_1, G_2)| - |mcs(G_1, G_2)|$
- A normalized dissimilarity is given by : $d_{MMCSN} = 1 - \frac{mcs(G_1,G_2)}{MCS(G_1,G_2)}$

Conclusion about this part

Exact graph matching

- Graph similarity can be evaluated using either graph isomorphims or common subgraphs
- Pro : a stringent definition and solid mathematical formulation
- Cons : require "exact" identity between graphs
  - for labels ($+10^{-8}$ on each label : no more similarity)
  - for structure

→ Too rigid for various real-world applications : how to uncompass ?

Outline of the lecture

1. Lecturer presentation
2. Metrics on graphs
3. Exact Graph Matching based similarities
   - Definitions and notations
   - Graph isomorphism and Weisfeiler-Lehman
   - Maximum common subgraph
4. Graph Edit Distance
   - Error-Tolerant graph Matching
   - Definitions and notations
   - Algorithms for exact GED computation
     - Tree-search algorithms

Rationale of Error-Tolerant graph Matching

Exact graph matching : pro and cons

- Pro : a stringent definition and solid mathematical formulation
- Cons : require "exact" identity between graphs, particularly for labels ($+10^{-8}$ on each label : no more similarity)

→ Too rigid for various real-world applications : how to uncompass ?

Error-Tolerant graph matching : principles

- Deal with more general matching problem
- Relaxes contraints and extend concept of isomorphisms :
  - Mapping of node $u \in V_1$ to node $f(u) \in V_2$ is possible even if $\mu_1(u) \neq \mu_2(f(u))$
  - Mapping of edge $e \in E_1$ to edge $f(e) \in E_2$ is possible even if $\zeta_1(u) \neq \zeta_2(f(u))$
  - Deletions (resp. Insertions) of nodes and edges are allowed

Error Tolerant Graph Matching (1)

Definition

General principle : Keep the previous definitions with a "Dummy Node" ϵ
Let $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$ be two graphs. An error-tolerant graph matching is a mapping $f : V_1 \cup \{\epsilon\} \to V_2 \cup \{\epsilon\}$. f is still bijective with respect to the real nodes but allows that several nodes from V_1 are simultaneously mapped to the dummy node, and reciprocally that ϵ can be mapped to several nodes from V_2. Formally :

- $f(u_1) \neq \epsilon \Rightarrow f(u_1) \neq f(u_2) \forall\, u_1, u_2 \in V_1$
- $f^{-1}(v_1) \neq \epsilon \Rightarrow f^{-1}(v_1) \neq f^{-1}(v_2) \forall\, v_1, v_2 \in V_2$

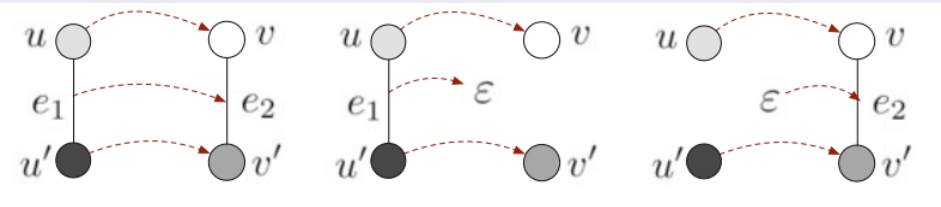
Each node of G_1 is either mapped to ϵ (deleted) or uniquely matched with a node of G_2.
Likewise, each node of G_2 is either mapped to ϵ (inserted) or uniquely matched with a node of G_1.

Error Tolerant Graph Matching (2)

Edge mapping in error-tolerant matching

Edges are processed in an error tolerant matching guided by the node mapping : an edge (u, v) is matched with an existing edge or with ϵ depending on the operation performed on u and v.
Let $u, u' \in V_1 \cup \{\epsilon\}$ and $v, v' \in V_2 \cup \{\epsilon\}$. Assume that both mappings $f(u) = v$ and $f(u') = v'$ are in the matching. 3 possible cases :

1. if there are edges $e_1 = (u, u') \in E_1$ and $e_2 = (v, v') \in E_2$, the edge mapping $e_1 \to e_2$ is implied by f
2. if there is an edge $e_1 = (u, u') \in E_1$ but no edge $e_2 = (v, v') \in E_2$, the edge deletion $e_1 \to \epsilon$ is implied by f
3. if there is no edge $e_1 = (u, u') \in E_1$ but an edge $e_2 = (v, v') \in E_2$, the edge insersion $\epsilon \to e_2$ is implied by f



Error Tolerant Graph Matching (3)

Error-tolerant matching as an optimization

- Question : how to derive a dissimilarity measure from an ETGM ?
- Answer : by assigning costs to the "tolerant" mappings with
  - high cost for dissimilar mapped nodes/edges and lower cost for similar
  - particular costs for insertions and deletions
- Thus an ETGM can be characterized by the overall cost of the mappings

This defines the principles of the Graph Edit Distance"

Do you remember the OR lecture from last year ?

Dynamic Programming

Minimum Edit Distance : Definition

The minimum edit distance between two strings is the minimum number of editing operations needed to transform a string into the other among

- insertion
- deletion
- substitution

Minimum Edit Distance : Applications

- Spell correction
  - The user typed "graffe"

  Which is closest?
  - graf
  - graft
  - grail
  - giraffe
- Computational Biology
  - Align two sequences of nucleotides

    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC
  - Resulting alignment:

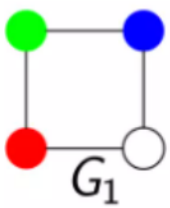
    −AGGCTATCACCTGACCTCCAGGCCGA−−TGCCC−−−
    TAG−CTATCAC−−GACCGC−−GGTCGATTTGCCCGAC

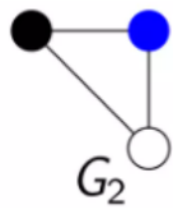
Graph-Edit distance (1)

Basic idea

Given two graphs, a source graph $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and a target graph $G_2 = (V_2, E_2, \mu_2, \zeta_2)$, the graph edit distance relies on the minimum cost of transforming G_1 into G_2 using edit operations on nodes and edges such as insertions, deletions and substitutions (possibly split and merge).

- $u \to v$ is the substitution of $u \in V_1$ and $v \in V_2$,
- $u \to \epsilon$ is the deletion of $u \in V_1$,
- $\epsilon \to v$ is the insertion of $v \in V_2$,
- The same notations exist for edges.



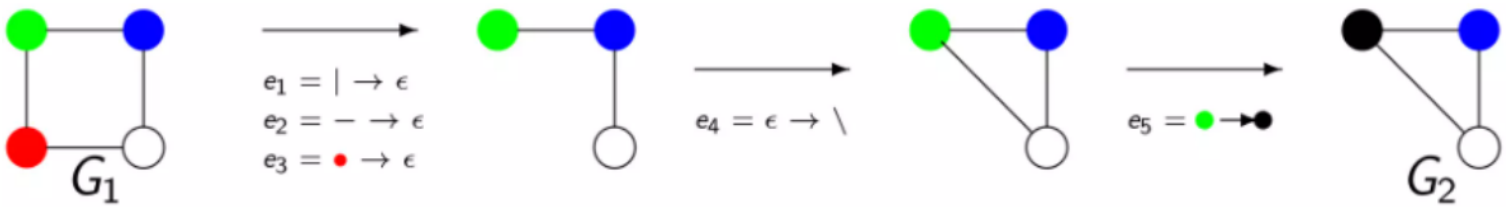


Graph-Edit distance (1)

Basic idea

Given two graphs, a source graph $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and a target graph $G_2 = (V_2, E_2, \mu_2, \zeta_2)$, the graph edit distance relies on the minimum cost of transforming G_1 into G_2 using edit operations on nodes and edges such as insertions, deletions and substitutions (possibly split and merge).

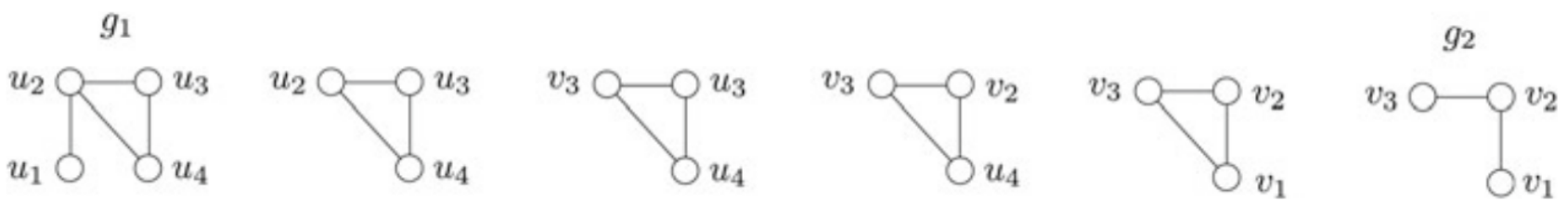
- $u \to v$ is the substitution of $u \in V_1$ and $v \in V_2$,
- $u \to \epsilon$ is the deletion of $u \in V_1$,
- $\epsilon \to v$ is the insertion of $v \in V_2$,
- The same notations exist for edges.



Graph-Edit distance (2)

Definition of an edit path

A set $\{e_1, e_2, ..., e_k\}$ of k edit operations e_i that transform completely a graph G_1 into a graph G_2 is called an edit path $\lambda(G_1, G_2)$ between G_1 and G_2. The set of all edit paths from G_1 to G_2 is denoted $\Gamma(G_1, G_2)$.
An edit path can be completely described using uniquely operations on nodes from V_1 and V_2. Edge edit operations are implicitely given by these node correspondences.
On the figure, the edit path $\lambda = \{(u_1 \to \epsilon), (u_2 \to v_3), (u_3 \to v_2), (u_4, v_1)\}$ corresponds to the edge edit operations $\{((u_1, u_2) \to \epsilon), ((u_2, u_3) \to (v_3, v_2))((u_3, u_4) \to (v_2, v_1))((u_2, u_4) \to \epsilon)$.



Graph-Edit distance (3)

Best Edit Path

- "All paths lead to Rome".
- Graph edit distance relies on the shortest one !
- In the GED framework : shortest = with the lowest cost
- The cost of an edit path is the sum of the operations $c(e_i)$

GED definition

$$GED(G_1, G_2) = \min_{\{e_1,\cdots,e_k\}\in\Gamma(G_1,G_2)} \sum_{i=1}^{k} c(e_i)$$

- $\Gamma(G_1, G_2)$ is the set of all edit paths from G_1 to G_2.
- $c(e_i)$ denotes the "strength" of node edit operation (including the cost of implied edges edit operations)
- The minimal cost edit path is not necessarily unique

Graph-Edit distance (4)

Conditions on edit cost functions

- $\Gamma(G_1, G_2)$ is infinite (e.g. by alternating $(u_1 \to \epsilon)$ and $(\epsilon \to u_1)$)
- By imposing some conditions to $c()$:
  - $c(e) \geq 0$ for all edit operations
  - $c(e) > 0$ for all insertion and deletion (which solves the problem above)
  - $c(u \to w) \leq c(u \to v) + c(v \to w)$
  - $c(u \to \epsilon) \leq c(u \to v) + c(v \to \epsilon)$
  - $c(\epsilon \to v) \leq c(\epsilon \to u) + c(u \to v)$

  $\to$ Superflous operations never decrease the cost.

With two more conditions :

- $c(e) = 0$ for identical substitutions
- $c(e) = c(e^{-1})$

GED becomes a metric

Graph-Edit distance (5)

Examples of edit costs

- For unlabelled graphs :
  - $c(u \to \epsilon) = c(\epsilon \to u') = c((u,v) \to \epsilon) = c(\epsilon \to (u',v')) = 1$
  - $c(u \to u') = c((u,v) \to (u',v')) = 0$
- For graphs labelled with vectors in $\mathbb{R}^n$:
  - $c(u \to \epsilon) = c(\epsilon \to v) = \tau$
  - $c(u \to v) = \left(\sum_{i=1}^{n} |\mu_1(u) - \mu_2(v)|^p\right)^{\frac{1}{p}}$ [the same for edges]

  Problem : A node substitution with a cost higher than 2τ is always replaced by a composition of deletion and insersion.
- More refined costs can be used for fitting application domains (for finite sets, for angles, for strings, or combinations of them)

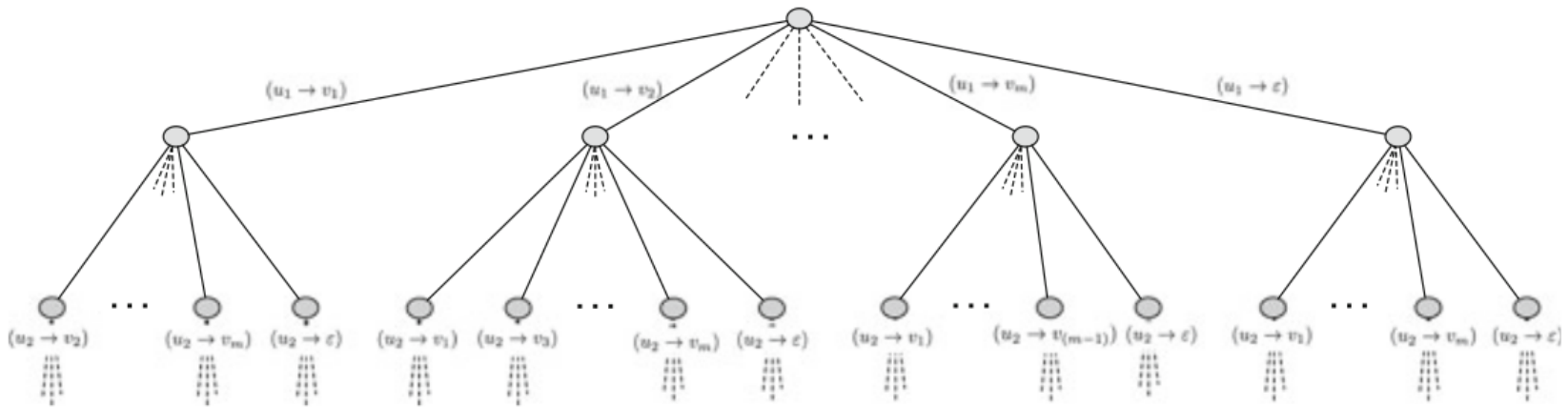
→ GED = very powerful tool, but very difficult to tune

Graph-Edit distance (6)

Another problem : the complexity of GED

Let $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$ be two graphs with respectively n and m nodes, i.e. $V_1 = \{u_1, ...u_n\}$ and $V_2 = \{v_1, ..., v_m\}$.

- If the matching is started with u_1, $(m+1)$ mappings are possible.
- Then, at the next step, m possibilities remain.
- Finally, this results in a complexity of $O(m^n)$



Outline of the lecture

Exact GED algorithms (1)

Basic Idea : tree-search

Most frequent approach :

- Tree search with heuristics to avoid the visit of unfruitful states.

Let us consider a partial edit path p and let :

- $g(p)$ be the cost of p and $h(p)$ be a Lower Bound (LB) of the cost of the remaining part of the path required to reach G_2

Since $h(p)$ is a LB : $d_\gamma(G_1, G_2) \geq g(p) + h(p) \ \forall \gamma = p|q \in \Gamma(G_1, G_2)$

Let UB denote the best approximation of the GED found so far.
If $g(p) + h(p) > UB$, we have $d_\gamma(G_1, G_2) \geq g(p) + h(p) > UB$

$\rightarrow$ All the sons of p will provide a greater approximation of the GED and correspond to unfruitful nodes. We can prune.

Exact GED algorithms (2)

How to choose h functions

Different choices are possible for function h [Abu-Aisheh, 2016].

- Null function : $h(p) = 0$
- Bipartite function (estimation of the true GED) : see next week.

Exact GED algorithms (3)

Example of partial tree search algorithm : A^*

Algorithm 1 Astar GED algorithm (*A*GED*)

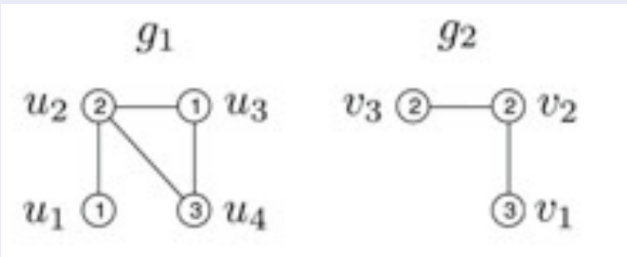
Input: Non-empty attributed graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ where $V_1 = \{u_1, ..., u_{|v_1|}\}$ and $V_2 = \{u_2, ..., u_{|v_2|}\}$

Output: A minimum cost edit path (p_{min}) from g_1 to g_2 *e.g.*, $\{u_1 \to v_3, u_2 \to \varepsilon, \varepsilon \to v_2\}$

1: $OPEN \leftarrow \{\phi\}$, $p_{min} \leftarrow \phi$
2: For each node $w \in V_2$, $OPEN \leftarrow OPEN \cup \{u_1 \to w\}$
3: $OPEN \leftarrow OPEN \cup \{u_1 \to \varepsilon\}$
4: **while** true **do**
5: $p_{min} \leftarrow \text{argmin}\{g(p) + lb(p)\}$ s.t. $p \in OPEN$
6: $OPEN \leftarrow OPEN \setminus p_{min}$
7: **if** p_{min} is a complete edit path **then**
8: Return p_{min} as a solution (*i.e.*, the minimum cost edit distance from g_1 to g_2)
9: **else**
10: Let $p_{min} \leftarrow \{u_1 \to v_{i1}, ..., u_k \to v_{ik}\}$
11: **if** $k < |V_1|$ **then**
12: For each $w \in V_2 \setminus \{v_{i1}, ..., v_{ik}\}$, $OPEN \leftarrow OPEN \cup \{p_{min} \cup \{u_{k+1} \to w\}\}$
13: $p_{new} \leftarrow p_{min} \cup \{u_{k+1} \to \varepsilon\}$
14: $OPEN \leftarrow OPEN \cup \{p_{new}\}$
15: **else**
16: $p_{new} \leftarrow p_{min} \cup \bigcup_{w \in V2 \setminus \{v_{i1}, ..., v_{ik}\}} \{\varepsilon \to w\}$
17: $OPEN \leftarrow OPEN \cup \{p_{new}\}$
18: **end if**
19: **end if**
20: **end while**

Let's take a "very small" example with

- L1 norm for node substitutions
- unary insertion/deletion costs
- unary edge substitution cost



Exact GED algorithms (3)

Example of partial tree search algorithm : A^*

Algorithm 1 Astar GED algorithm (*A*GED*)

Input: Non-empty attributed graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ where $V_1 = \{u_1, ..., u_{|v_1|}\}$ and $V_2 = \{u_2, ..., u_{|v_2|}\}$

Output: A minimum cost edit path (p_{min}) from g_1 to g_2 *e.g.*, $\{u_1 \to v_3, u_2 \to \varepsilon, \varepsilon \to v_2\}$

1: $OPEN \leftarrow \{\phi\}$, $p_{min} \leftarrow \phi$
2: For each node $w \in V_2$, $OPEN \leftarrow OPEN \cup \{u_1 \to w\}$
3: $OPEN \leftarrow OPEN \cup \{u_1 \to \varepsilon\}$
4: **while** true **do**
5: $p_{min} \leftarrow \text{argmin}\{g(p) + lb(p)\}$ s.t. $p \in OPEN$
6: $OPEN \leftarrow OPEN \setminus p_{min}$
7: **if** p_{min} is a complete edit path **then**
8: Return p_{min} as a solution (*i.e.*, the minimum cost edit distance from g_1 to g_2)
9: **else**
10: Let $p_{min} \leftarrow \{u_1 \to v_{i1}, ..., u_k \to v_{ik}\}$
11: **if** $k < |V_1|$ **then**
12: For each $w \in V_2 \setminus \{v_{i1}, ..., v_{ik}\}$, $OPEN \leftarrow OPEN \cup \{p_{min} \cup \{u_{k+1} \to w\}\}$
13: $p_{new} \leftarrow p_{min} \cup \{u_{k+1} \to \varepsilon\}$
14: $OPEN \leftarrow OPEN \cup \{p_{new}\}$
15: **else**
16: $p_{new} \leftarrow p_{min} \cup \bigcup_{w \in V2 \setminus \{v_{i1}, ..., v_{ik}\}} \{\varepsilon \to w\}$
17: $OPEN \leftarrow OPEN \cup \{p_{new}\}$
18: **end if**
19: **end if**
20: **end while**

Properties of A^*

- If it terminates, it returns the optimal value of the GED
- The set OPEN can be as large as the number of edit paths between G_1 and G_2
- It does not return any result before it finds the optimal solution

Exact GED algorithms (4)

Other example of tree search algorithm : Depth First Search

Input: Two graphs G_1 and G_2 with $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$
Output: γ_{UB} and UB a minimum edit path and its associated cost

$(\gamma_{UB}, UBCOST) = GoodHeuristic(G_1, G_2)$
initialize $OPEN$
while $OPEN \neq \emptyset$ **do**
    $p = OPEN.popFirst()$
    if p is a leaf (i.e. if all vertices of V_1 are mapped) **then**
        complete p by inserting pending vertices of V_2
        update $(\gamma_{UB}, UBCOST)$ if required
    else
        Stack into OPEN all sons q of p such that $g(q) + h(q) < UBCOST$.
    end if
end while

Properties of DFS

- The number of pending edit path in open is bounded by $|V_1||V_2|$
- The initialization using an heuristic allow to discard many branches
- The algorithm quickly find a first edit path, it can be distributed