Quelle IA pour les données structurées ?

Romain Raveaux

romain.raveaux@univ-tours.fr

Maître de conférences

Université de Tours

Laboratoire d'informatique (LIFAT)

Equipe RFAI

Outline

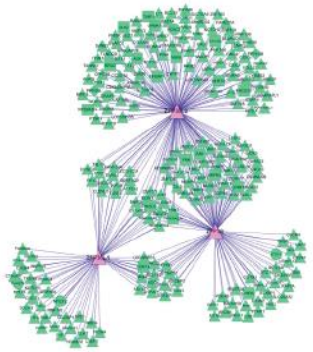
- Type of data
  - Euclidean data
  - Structured data
- Graph-based problems
  - Graph classification
  - Node classification
- Methods
  - Graph embedding:
    - Graph kernels
    - Graph neural networks

Type of Data

Text

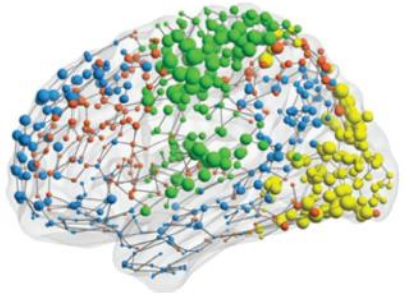
Audio signals

Social networks

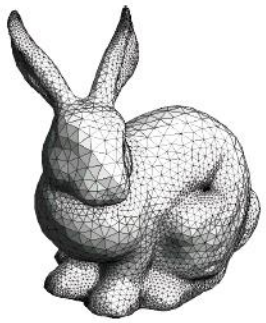


Regulatory networks

Images



Functional networks



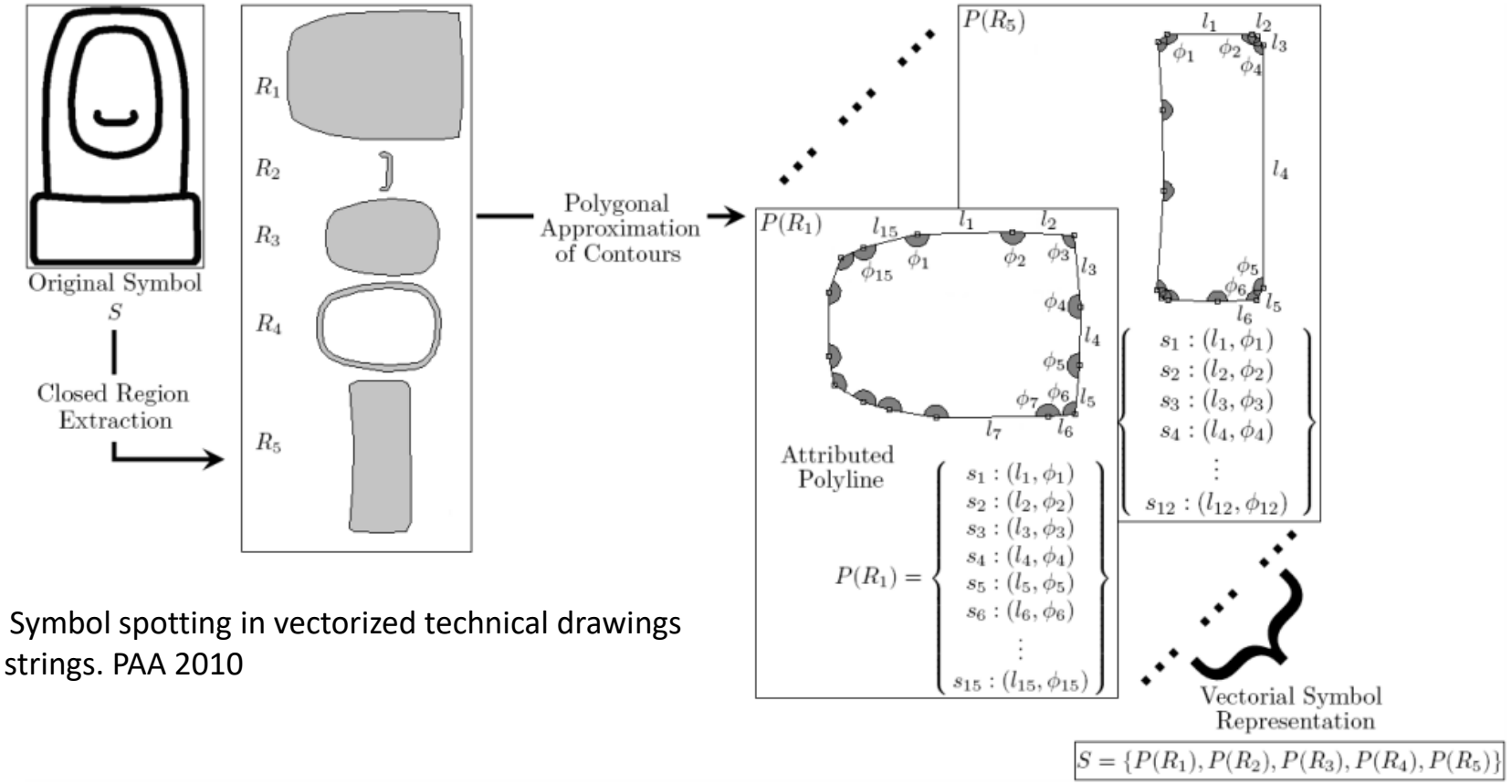
3D shapes

Taken from M. Bronstein. CVPR Tutorial 2017

Structured Data

- String
- Tree
- Graph

String

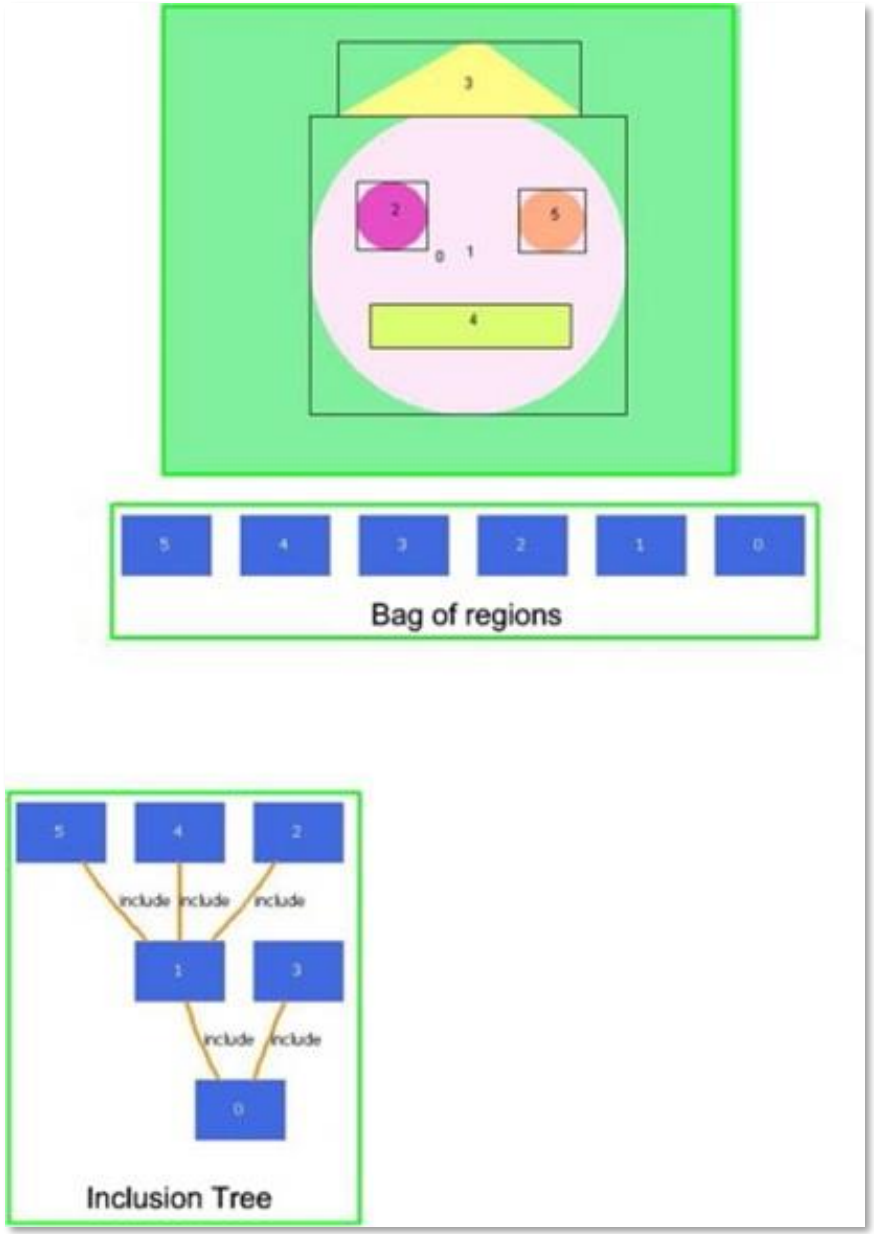


Taken from: Marçal Rusiñol et al: Symbol spotting in vectorized technical drawings through a lookup table of region strings. PAA 2010

X: A quote from Mahatma Gandhi: “Be the change you wish to see in the world”

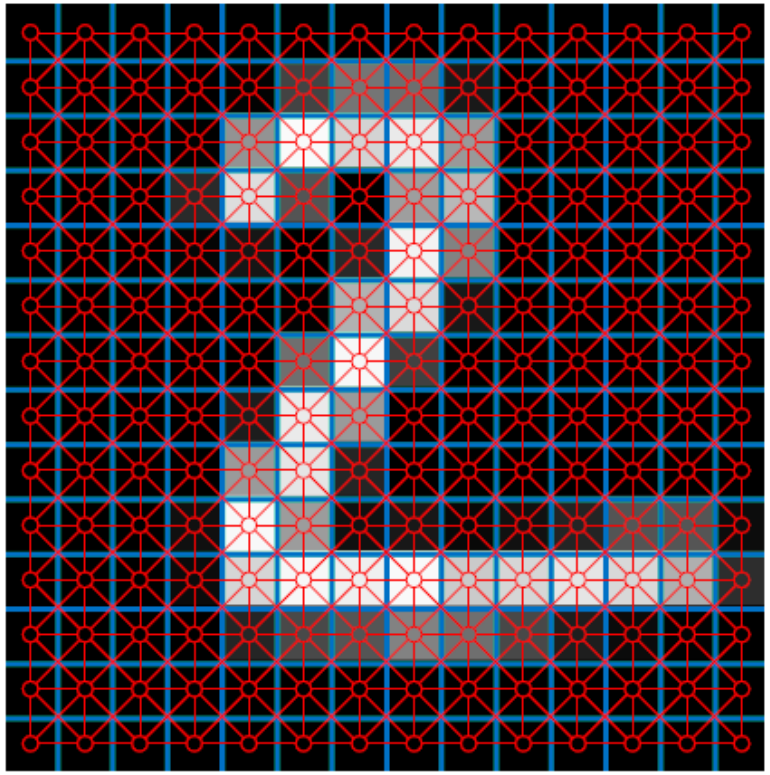
$X^{<1>}$ $X^{<2>}$ $X^{<3>}$ ..... $X^{<18>}$

Tree

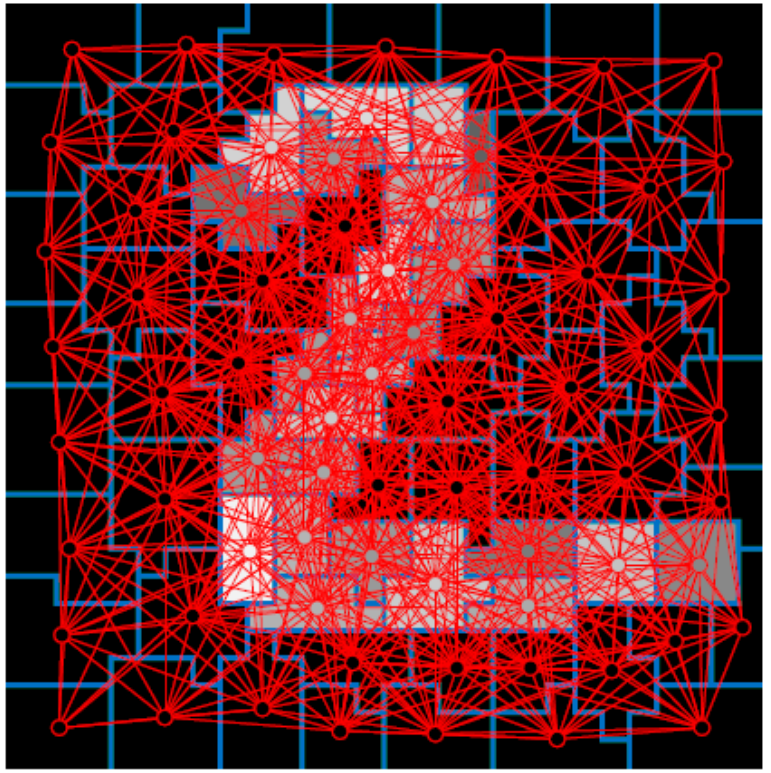


R. Raveaux et al:
Structured representations in a content based image retrieval context. J. Visual Communication and Image Representation 24(8): 1252-1268 (2013)

Graph

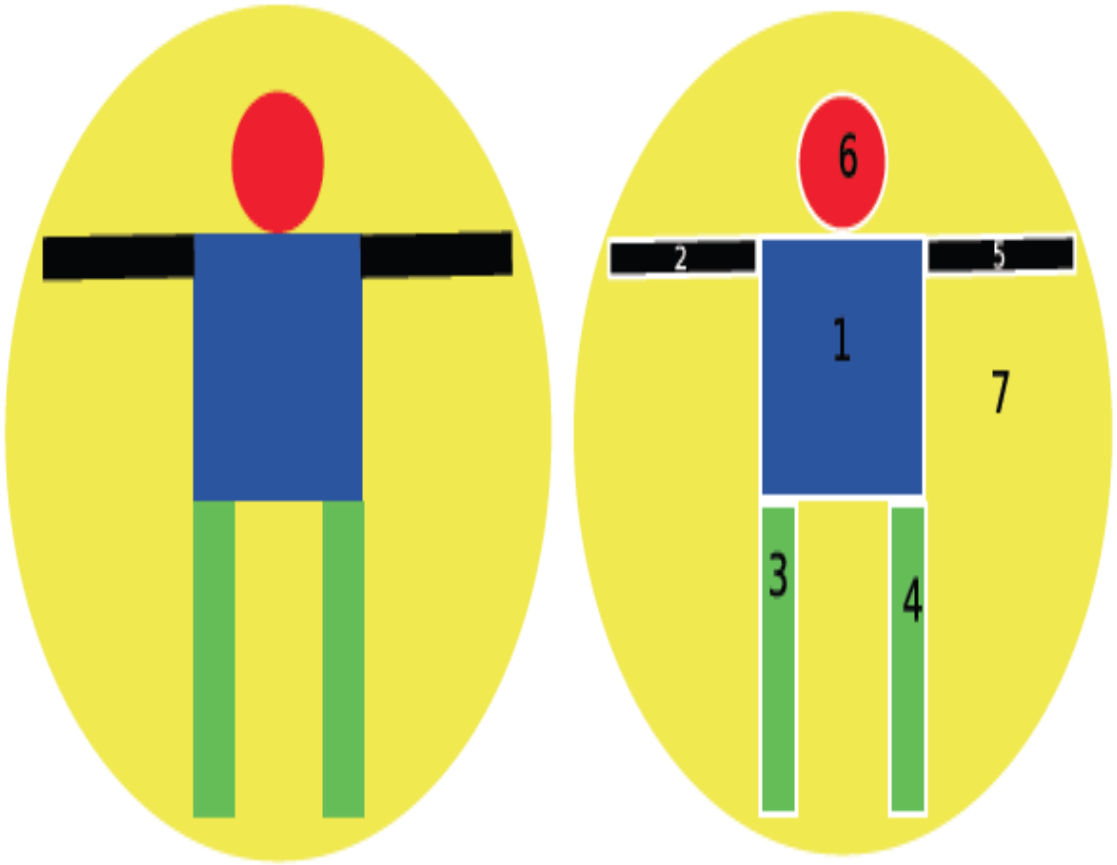


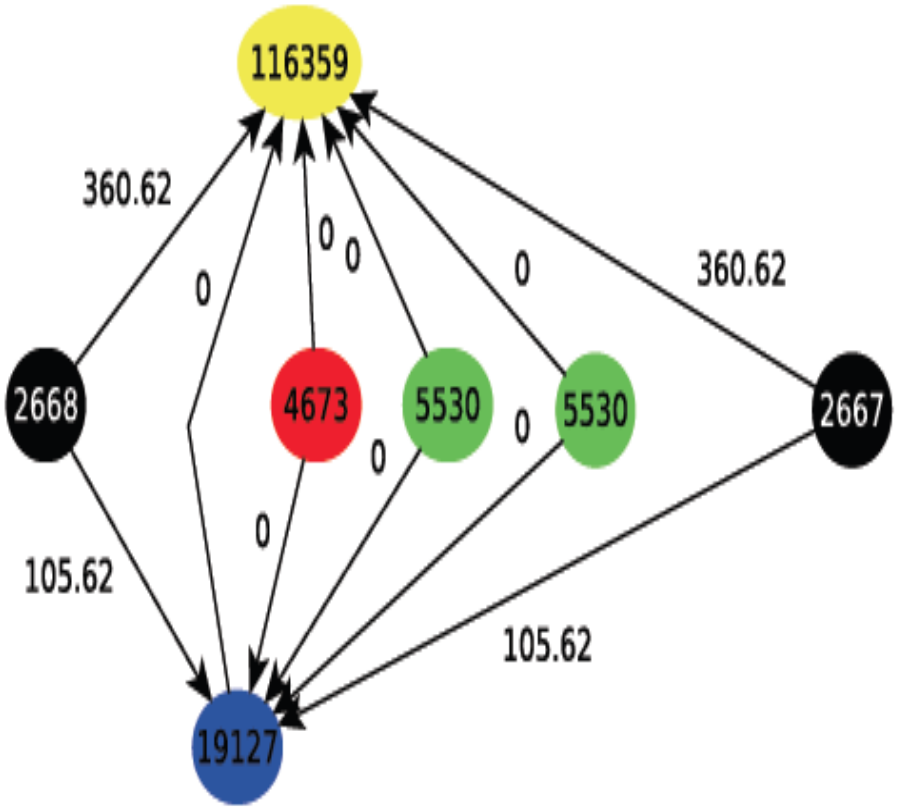
Regular grid



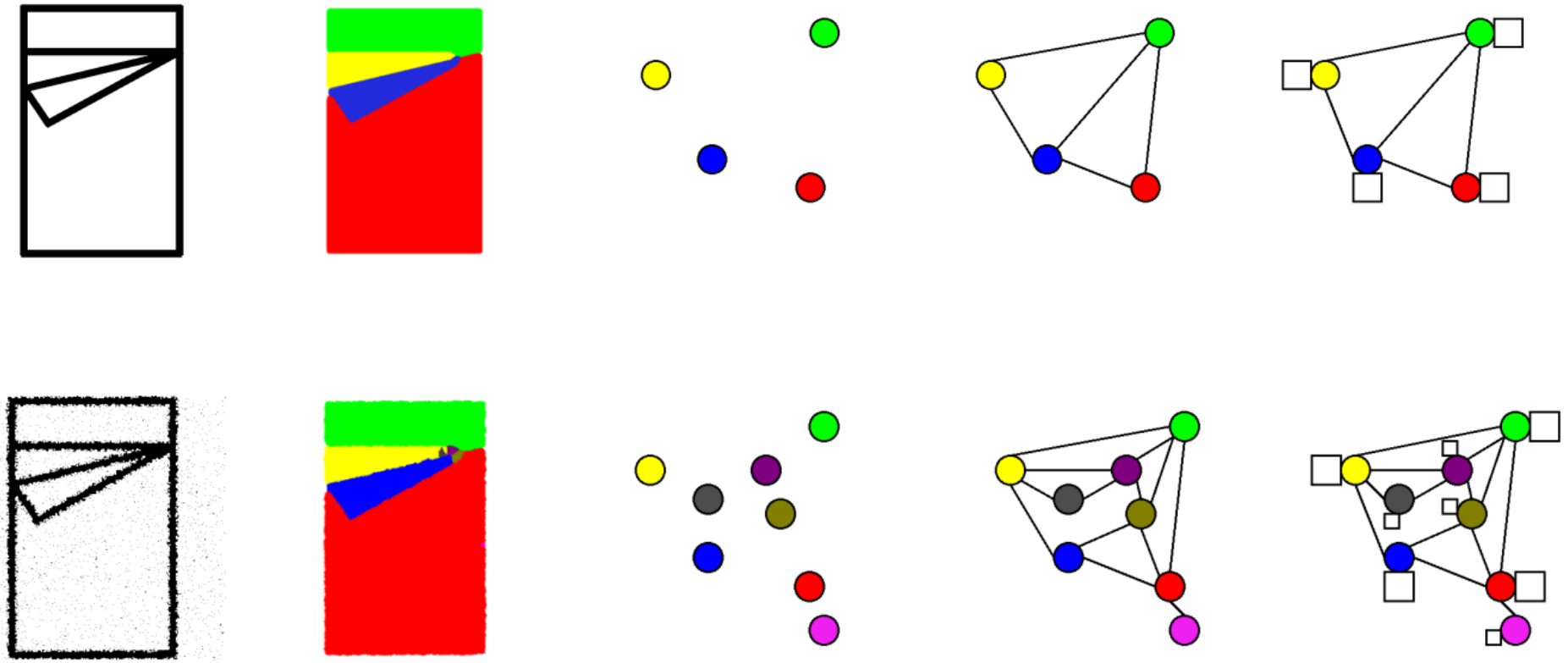
Superpixels

Adjacency Graph



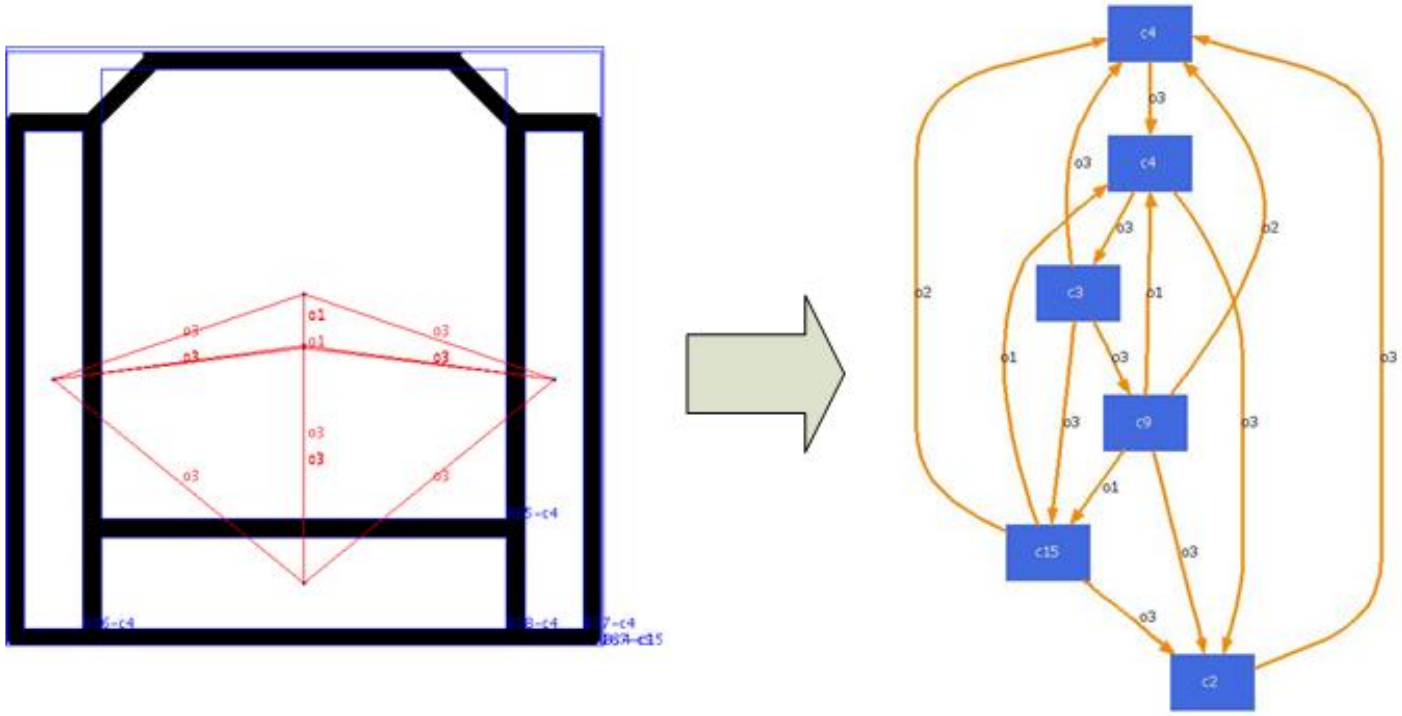


Region Adjacency Graph

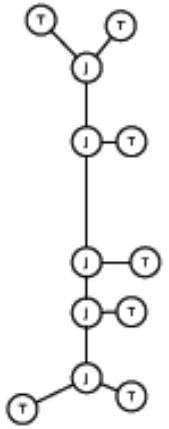


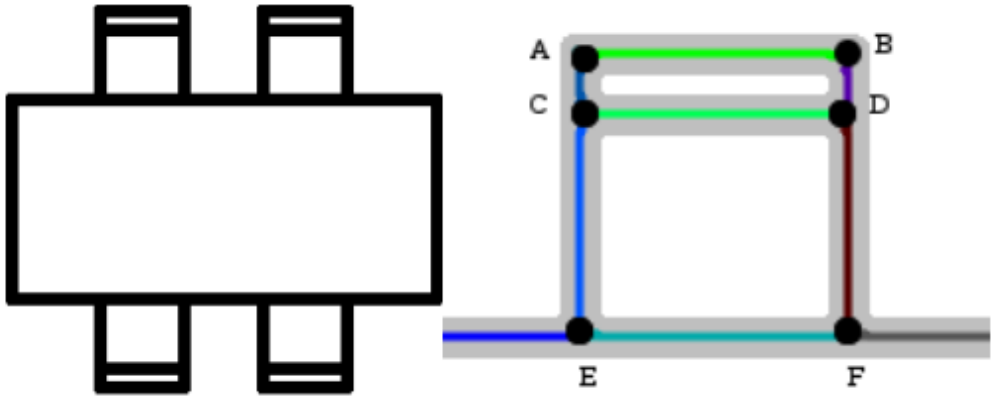
Impact of noise on Graph-Based Representation

Neighborhood graph

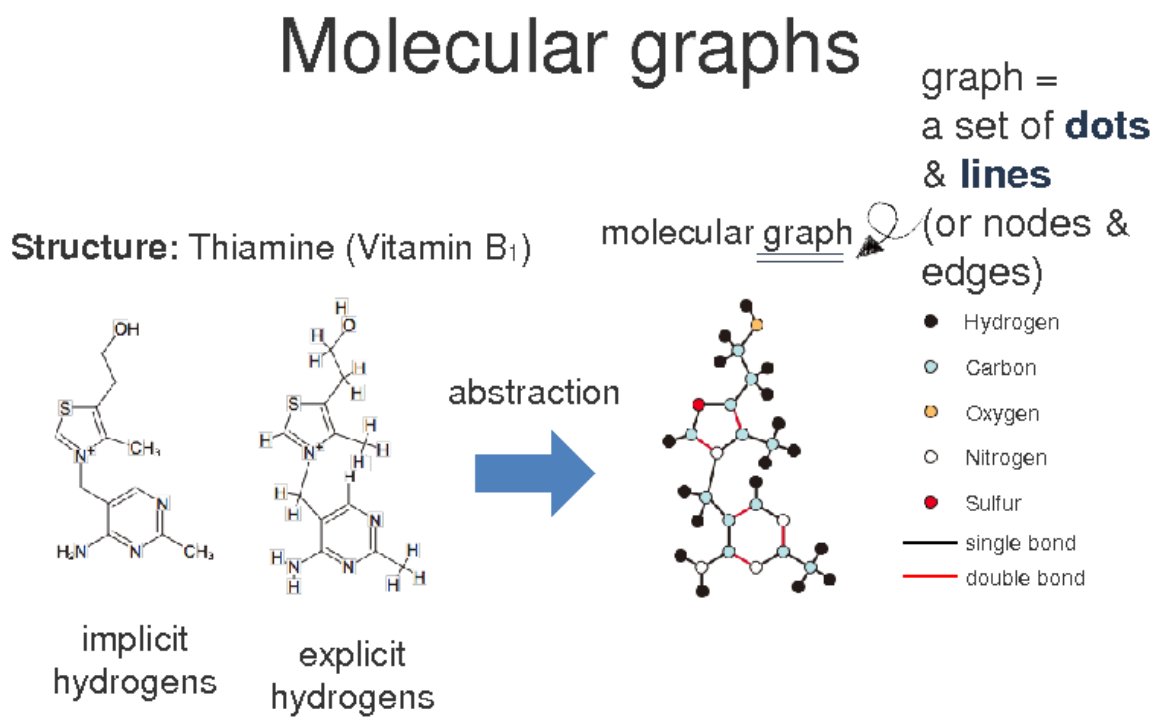


Skeleton Graph

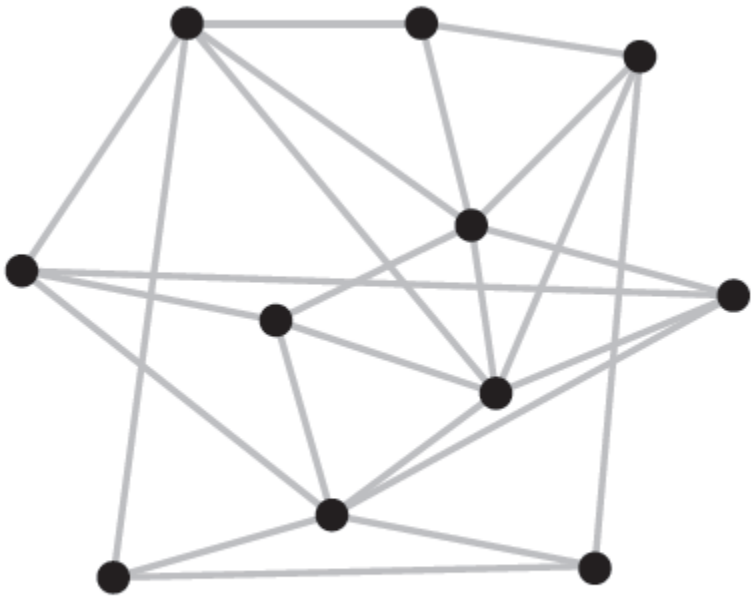




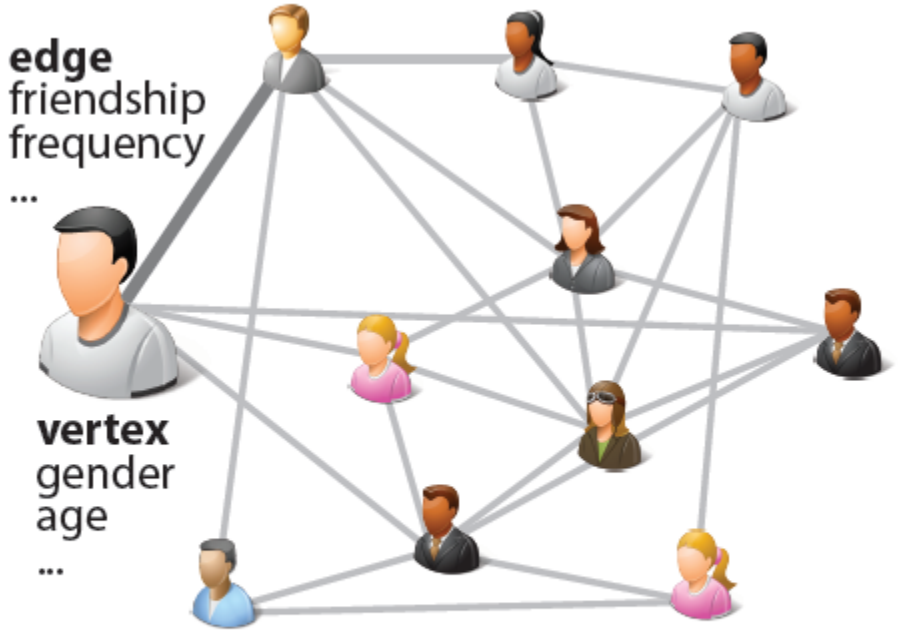
Graph of molecule : Chemoinformatics



Domain structure vs Data on a domain



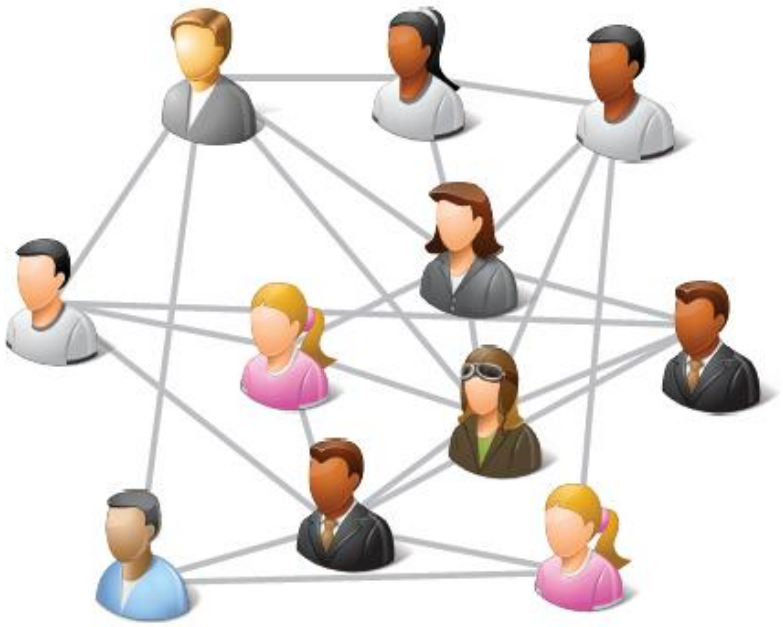
Domain structure



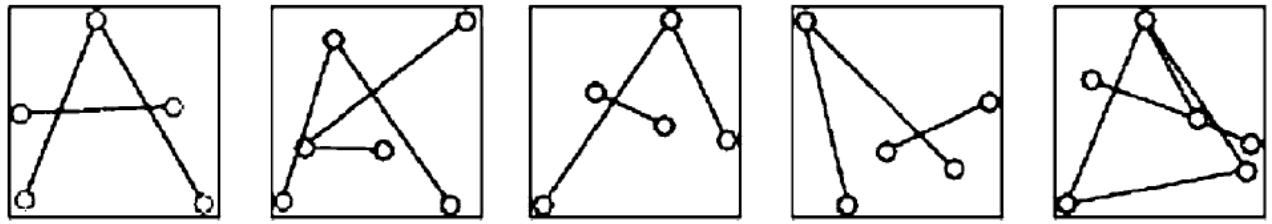
Data on a domain

Taken from M. Bronstein. CVPR Tutorial 2017

Fixed vs different domain



Social network
(fixed graph)



1D,2D, 3D shapes
(Different graphs)

Taken from M. Bronstein. CVPR Tutorial 2017

Structured data

- We will focus on graphs:
  - Graph as a generalization of Euclidean data : vector, matrix, tensors ….
  - Graphs as a generalization of strings and trees.
  - By nature, data are more likely to be graphs.

Graph-based problems

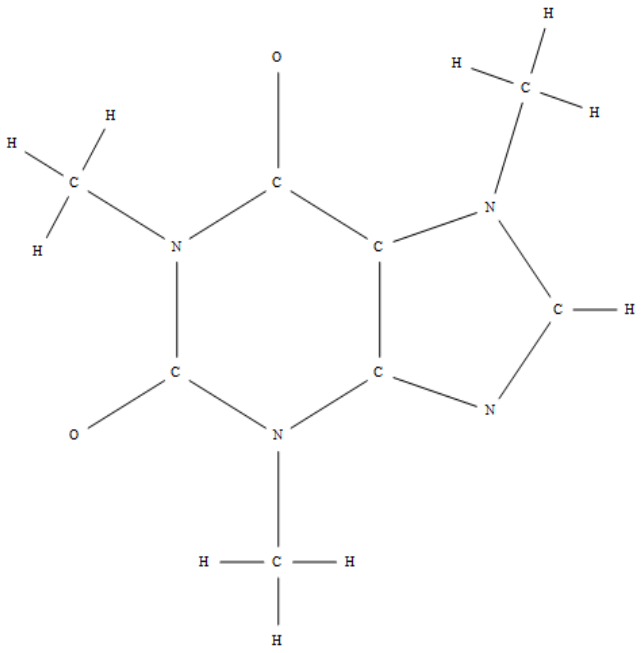
- Graph classification/clustering/regression
- Vertex classification/clustering/regression
- Graph matching
- Graph distance
- Graph-based search
  - Subgraph search
  - Subgraph spotting
  - Similarity search
- Graph prototypes
  - Median graphs/ Super graphs

IA is the set of tools to solve these problems : Modelisation, Machine learning, Optimization … …

Graph classification

1. cancerous or not cancerous molecules

2. determination of the boiling point

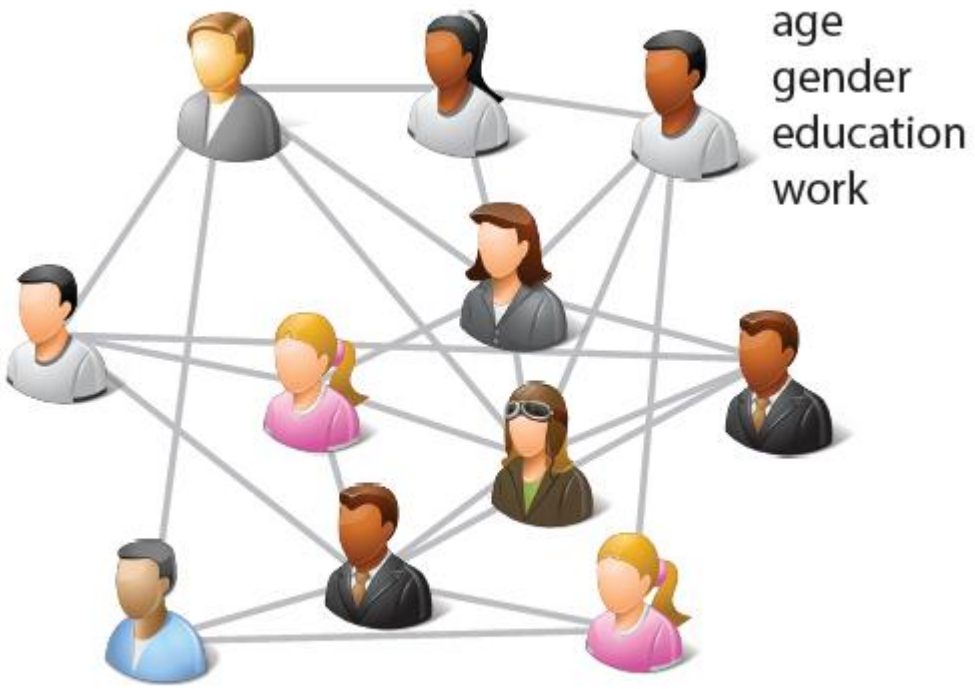


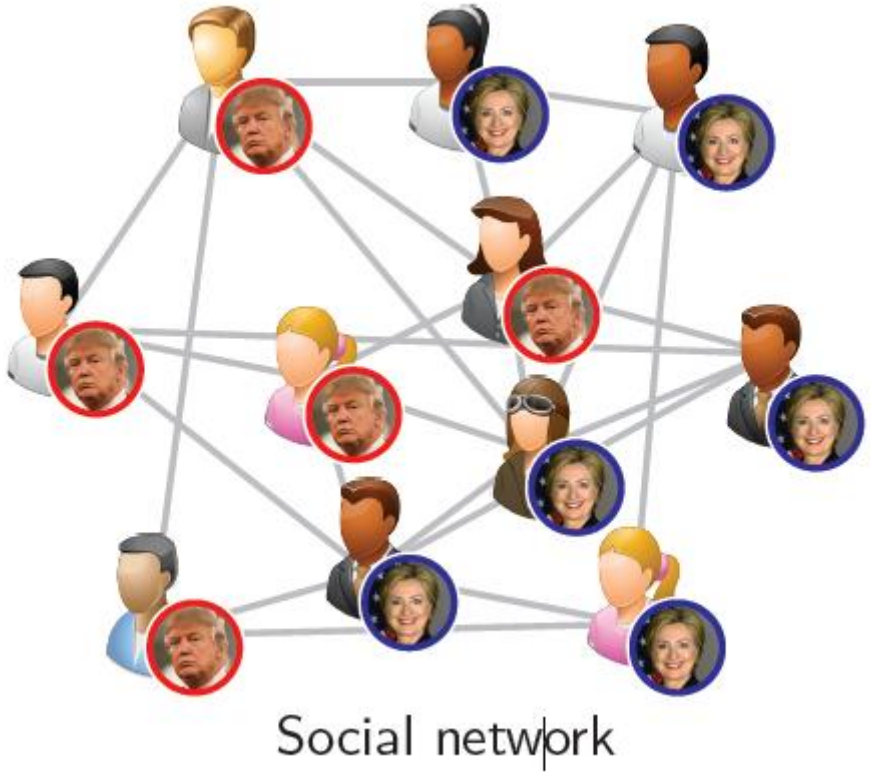
Molecular graph

Image recognition

Taken from M. Bronstein. CVPR Tutorial 2017

Vertex classification

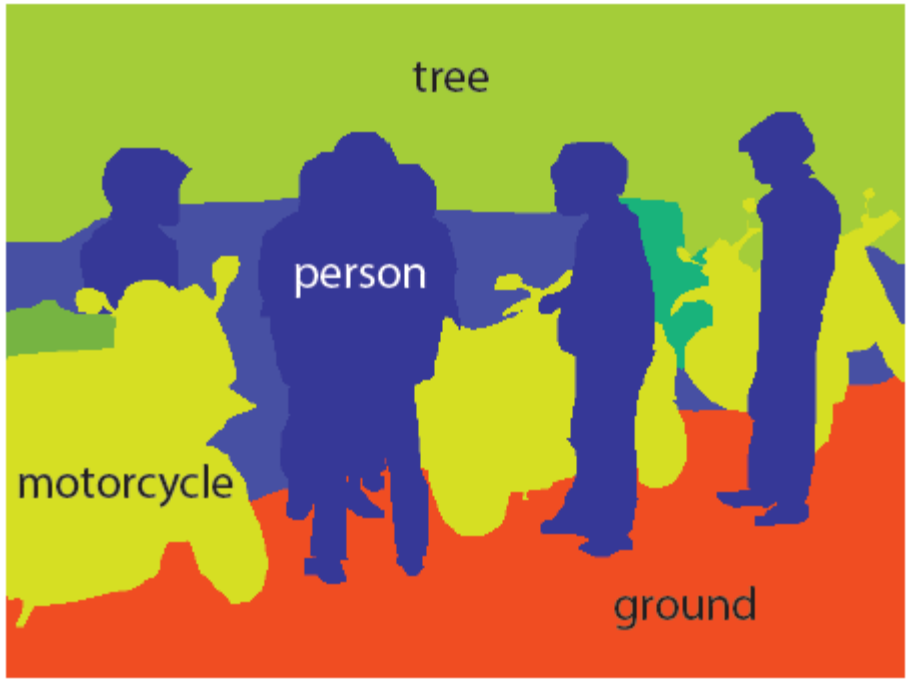




Taken from M. Bronstein. CVPR Tutorial 2017

Vertex classification/clustering

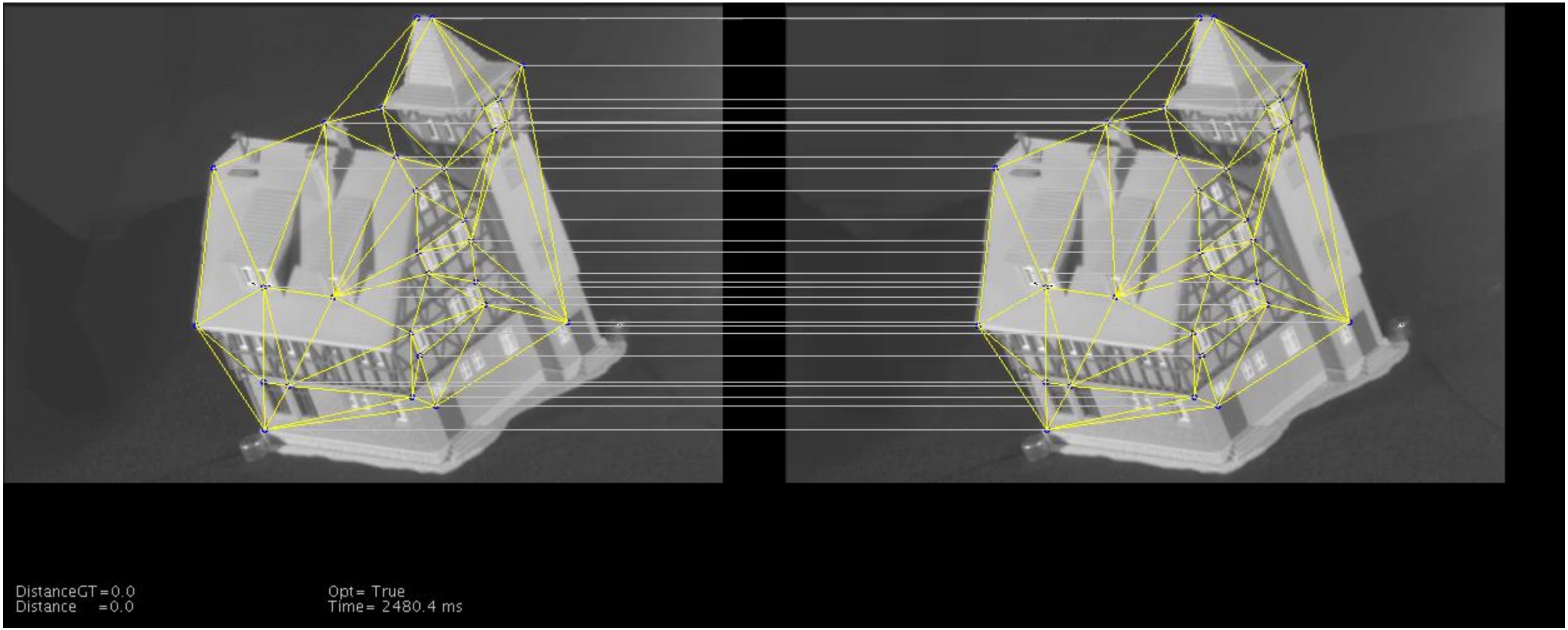
Semantic image segmentation



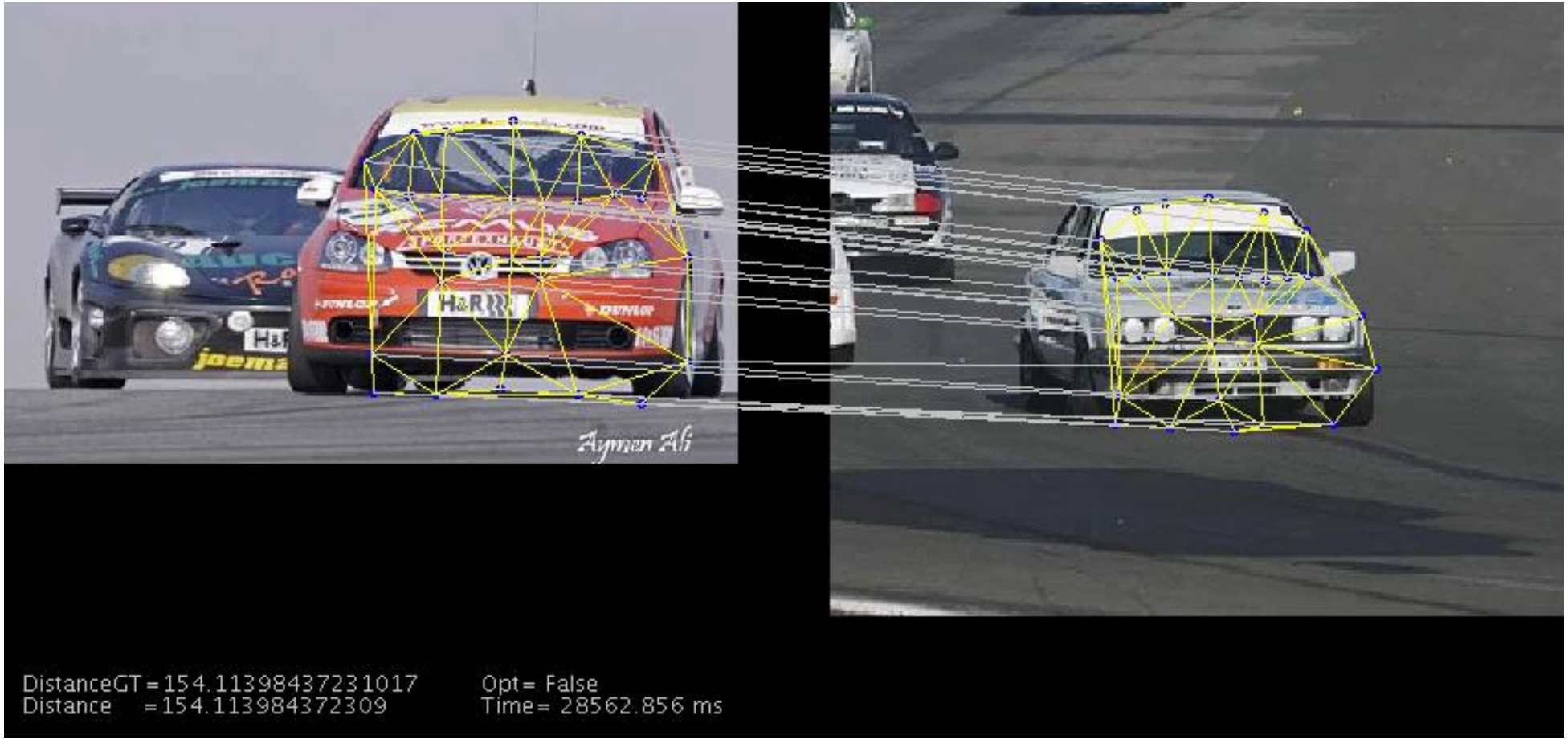
Semantic image segmentation

Taken from M. Bronstein. CVPR Tutorial 2017

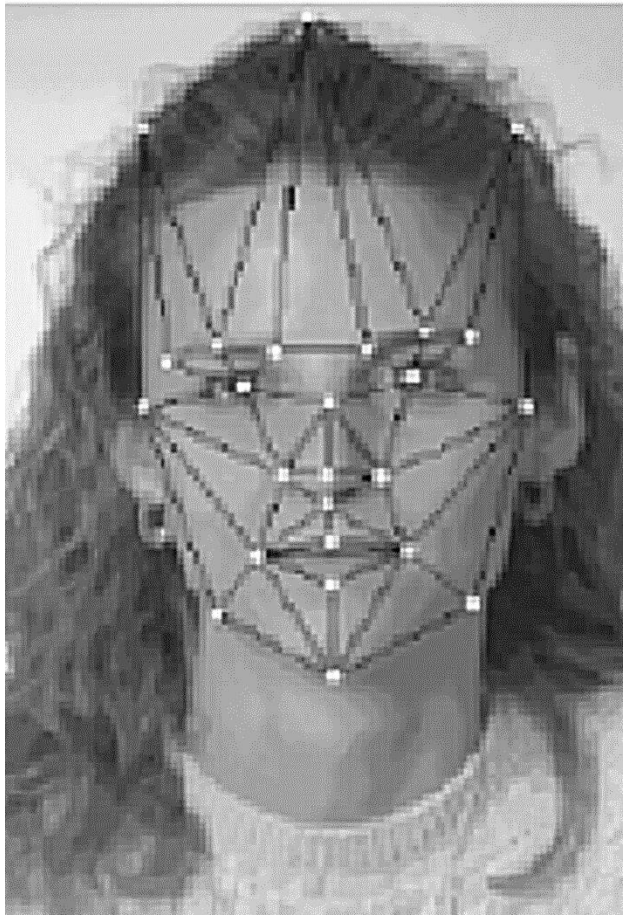
Graph matching



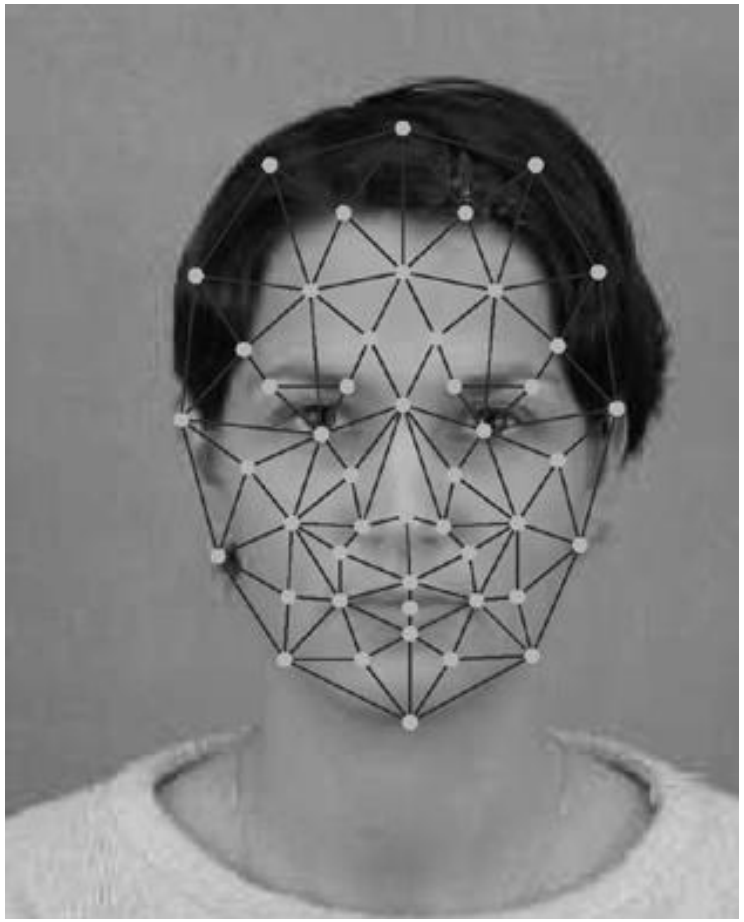
Graph matching



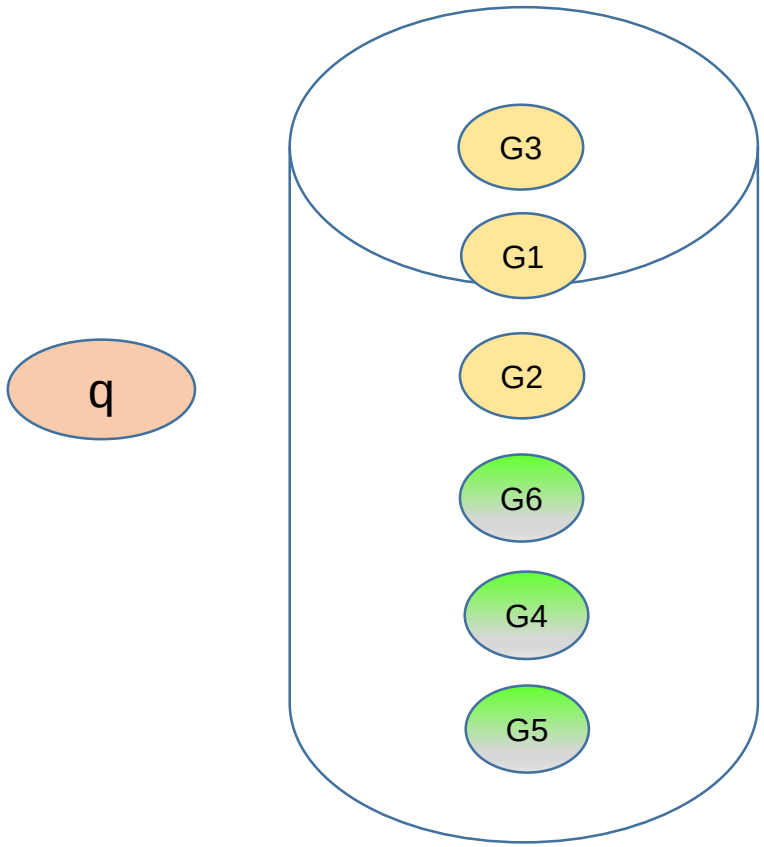
Graph distance



How similar are theses graphs ?

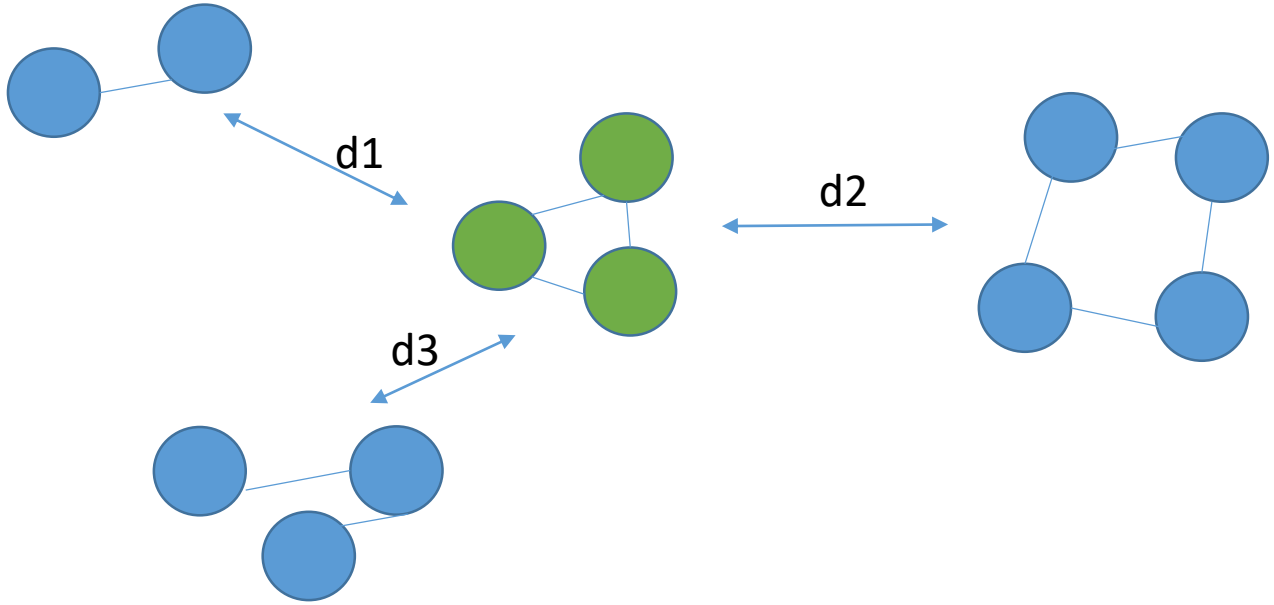


Graph-based search



1. Given a graph database consisting of n graphs, $D = g1, g2, ..., gn$, and a query graph q, almost all existing algorithms of processing graph search can be classified into the following four categories: Full graph search, Subgraph search, Similarity search and Graph mining.
2. **Full graph search.** Find all graphs gi in D s.t. gi is the same as q.
3. **Subgraph search**. Find all graphs gi in D containing q or contained by q.
4. **Similarity search**. Find all graphs gi in D s.t. gi is similar to q within a user-specified threshold based on some similarity measures.
5. **Graph mining** Graph mining problem gathers similar graph or subgraph of D in order to find clusters or prototypes. No query is provided by the user.

Median graph



What do we need to solve these problems ?

- Modelisation, Machine learning, Optimization, …:
- Graph space:
  - Graph matching
    - Combinatorial problems (NP-Hard)
- Graph embedding: $G \to \mathbb{R}^n$
  - Embedded of graphs/nodes/edges into a vector space.
    - Explicit embedding:
      - Through feature extraction (handcrafted or end-to-end learning) or dissimilarities
    - Implicit embedding:
      - Through graph kernel

Graph Neural Network

Taxonomy:

- Graph/node embedding
  - Explicit embedding
    - Through feature extraction
      - End-to-end learning

Graph Neural Network

- Input: A graph
- Output: Node embeddings
- Assumptions: stationarity and compositionality
- The goal:
  - Graph Neural Networks (GNN) perform an end-to-end learning including feature extraction and classification.

The basics of artificial neural networks

Let $x \in \mathbb{R}^{1 \times m}$ be a vector considered as an input data.

$$H^{(l+1)} = f(H^{(l)})$$

$$H^{(l+1)} = \sigma(H^{(l)} W^{(l)}) \quad \forall l > 0$$

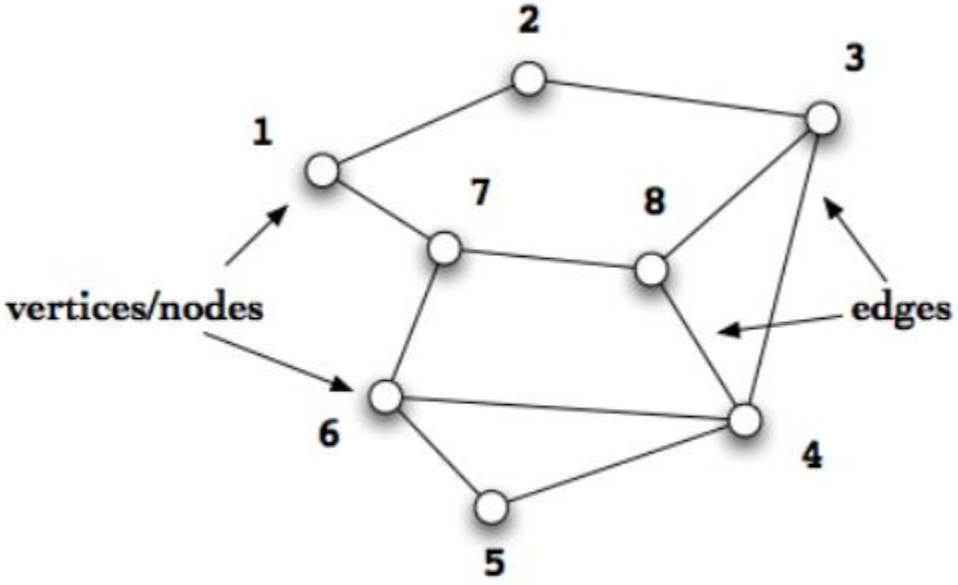
$W^{(l)} \in \mathbb{R}^{m_l \times m_{l+1}}$ is a matrix of trainable parameters. m_l is the number of neurons of the layer l. For the layer 0, $H^{(0)} = x$. Layer $l+1$ produces a vector $H^{(l+1)} \in \mathbb{R}^{1 \times m_{l+1}}$. Finally, σ is a non linear function. This neural network is considered as a model where parameters can be learned. This model is also denoted as a *"dense" layer* or *"Fully Connected (FC)" layer* or a *"MuLtilayer Perceptron"* (MLP). The question is how to generalize this artificial neural networks to graphs? What to do when the input is a graph?

The basics of graph neural networks

Definitions Assume we have a graph G:

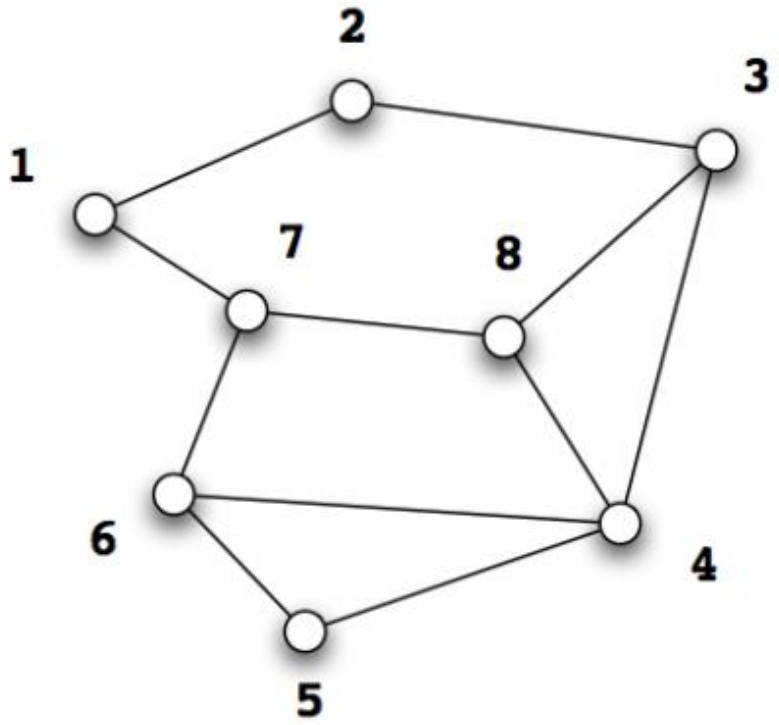
- V is the vertex set.
- E is the edge set.
- A is the adjacency matrix (assume binary). $A \in \{0,1\}^{|V| \times |V|}$
- F $\in \mathbb{R}^{|V| \times m}$ is a matrix of node features.
  - Categorical attributes, text, image data
  - Node degrees, clustering coefficients, etc.
  - Indicator vectors (i.e., one-hot encoding of each node)

Adjacency matrix



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Degree matrix



$$D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

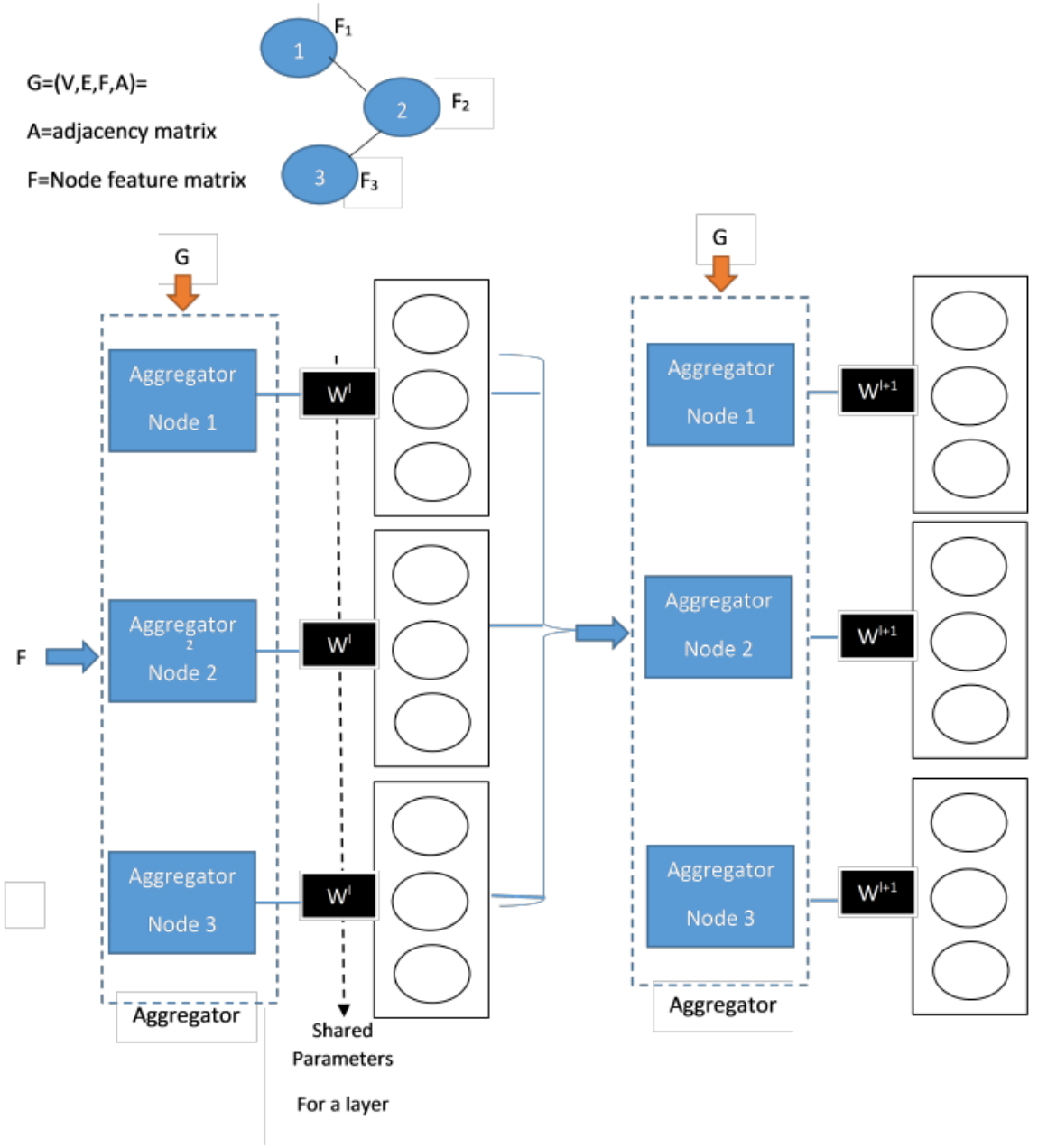
Normalized Adjacency matrix $\tilde{A} = D^{-1}A$ is a stochastic matrix (each row sums to one)

Key idea and Intuition [Kipf and Welling, 2016]

- The key idea is to generate node embeddings based on local neighborhoods.
- The intuition is to aggregate node information from their neighbors using neural networks.
- Nodes have embeddings at each layer and the neural network can be arbitrary depth. “layer-0” embedding of node u is its input feature, i.e. Fu.

$$H^{(l+1)} = f(H^{(l)}, A)$$

GNN: a pictorial model



Simple example of f

$$f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right)$$

where $W^{(l)} \in \mathbb{R}^{m_l \times m_{l+1}}$ is a weight matrix for the l-th neural network layer. m_l is indexed by l because it depends on the number of parameters in the layer l. Theses values are hyperparameters of the model except for $H^{(0)} = F$. $\sigma(.)$ is a non-linear activation function like the ReLU. Note that $\sigma(.)$ is a element-wise non-linearity operating on a matrix. But first, let us address two limitations

2 issues of this simple example

- Issue 1:
  - for every node, f sums up all the feature vectors of all neighboring nodes but not the node itself.
  - Fix: simply add the identity matrix to A → $\hat{A} = A + I$
- Issue 2:
  - A is typically not normalized and therefore the multiplication with A will completely change the scale of the feature vectors.
  - Fix: Normalizing A such that all rows sum to one: $D^{-1}A$

$$f(H^{(l)}, A) = \sigma\left(D^{-1}AH^{(l)}W^{(l)}\right)$$

Altogether: [Kipf and Welling, 2016]

- The two patched mentioned before +
- A better (symmetric) normalization of the adjacency matrix

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

More complexe models [Nowak et al., 2017]

- $\mathbf{I} \in \mathbb{R}^{|V|\times|V|}$. This identity operator does not consider the structure of the graph and neither provide any aggregation. Used alone this operator makes the GNN a composition of $|V|$ MLP completly independent. One MLP for each node feature vector.
- $A \in \mathbb{R}^{|V|\times|V|}$. The adjacency operator gather information on the node neighborhood (1 hop).
- $D \in \mathbb{R}^{|V|\times|V|}$. $D = diag(A\mathbf{1}$. This degree operator gather information on the node degree. D is node degree matrix (a diagonal matrix).
- $A_j \in \mathbb{R}^{|V|\times|V|}$. $A_j = min(1, A^{2^j})$. It encodes 2^j-hop neighborhoods of each node, and allow us to aggregate local information at different scales, which is useful in regular graphs.
- U $\in 1^{|V|\times|V|}$. U is matrix filled with ones. This average operator, which allows to broadcast information globally at each layer, thus giving the GNN the ability to recover average degrees, or more generally moments of local graph properties.

By denoting $\mathcal{A} = \{\mathbf{1}, D, A, A_1, \cdots, A_J, U\}$, a GNN layer is defined as :

$$f(H^{(l)}, \mathcal{A}) = \sigma\left(\sum_{B\in\mathcal{A}} BH^{(l)}W_B^{(l)}\right)$$

$\Omega = \{W_1^{(l)}, \cdots, W_{|\mathcal{A}|}^{(l)}\}$, $W_B^{(l)} \in \mathbb{R}^{m_{(l)}\times m_{(l+1)}}$ are trainable parameters. All the nodes share the same operators but it is not mandatory.

GNN as an encoder

$$ENC : G \rightarrow \mathbb{R}^{|V| \times p}$$

Applications and losses

- Let us recall that Z is the output the GNN

Unsupervised learning

- The graph factorization problem is the problem of predicting if two nodes are linked or not.

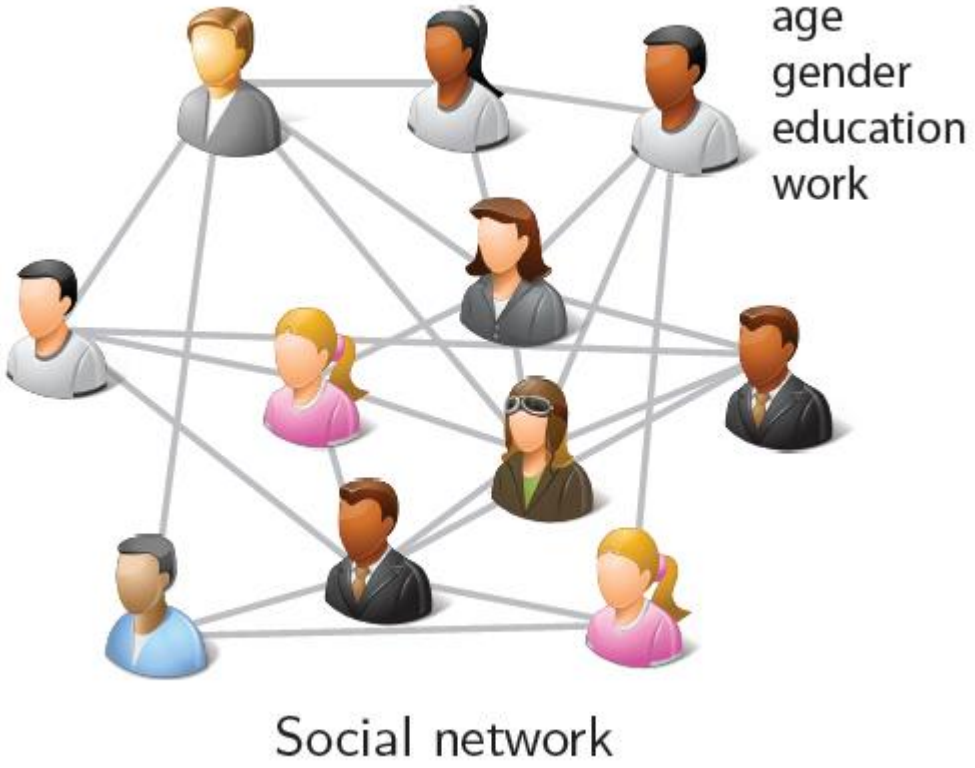
$$l = \sum_{(u_i,u_j)\in\mathcal{D}} (Z_i^T Z_j - A_{i,j})^2$$

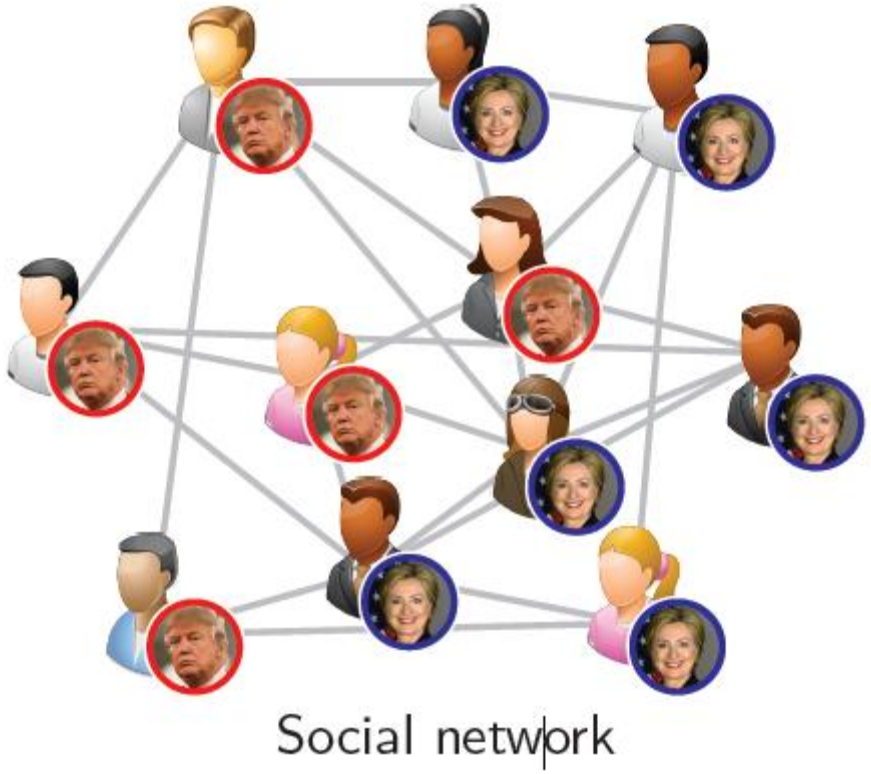
- Where $Z_i^T Z_j$ is a similarity measure between two nodes embeddings

- Variation of graph factorization
  - DeepWalk [Perozzi et al., 2014]
  - node2vec [Grover and Leskovec, 2016]

Supervised learning

- Node classification : social network

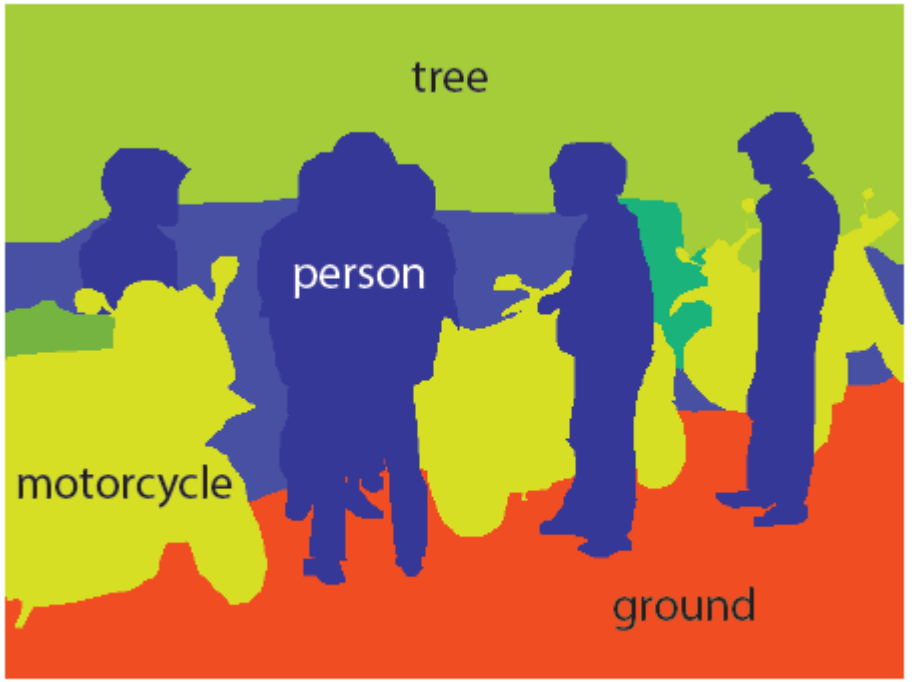




Supervised learning

- Node classification :

Semantic image segmentation



Semantic image segmentation

Supervised learning :Node classification

- Last layer:

$$Z = \mathbb{R}^{|V| \times p}$$

- For the last layer the activation function is a softmax activation function.

$$softmax(Z_{i,j}) = \frac{\exp(Z_{i,j})}{\sum_{i=1}^{p} \exp(Z_{i,j})}$$

- The loss :

$$l = -\sum_{(u_i) \in \mathcal{D}} \sum_{j=1}^{p} t_{i,j} \log(Z_{i,j})$$

Where $t_i \in \{0, 1\}^p$ is a one-hot vector of the ground-truth class label for node u_i and $\mathcal{D}$ is the data set composed of nodes.

Supervised learning:

- Graph classification
- A global average pooling layer must be added to gather all the node embeddings of a given graph.

$$Z' = \mathbb{R}^{1 \times p} \qquad Z'_{1,j} = \frac{1}{|V|} \sum_{i=1}^{|V|} Z'_{i,j} \quad \forall j \in 1, \cdots, p.$$

- Z' can be fed to a MLP for classification
- The loss is the cross entropy

$$l = -\sum_{(G_i) \in \mathcal{D}} \sum_{j=1}^{\#classes} t_{i,j} \log(\hat{t}_{i,j})$$

Where $\#classes$ is the number of classes. $\mathcal{D}$ is the data set composed of graphs. $t_i \in \{0,1\}^{\#classes}$ is a one-hot vector of the ground-truth class label for graph G_i. Mettre figure.

Semi-Supervised learning:

- Node classification:
  - the problem of classifying nodes in a graph, where labels are only available for a small subset of nodes.
  - where label information is smoothed over the graph via some form of explicit graph-based regularization
  - Assumption is that connected nodes in the graph are likely to share the same label (class).
    - This is true for instance for a neighborhood graph.

$$l = l_0 + \lambda l_{reg}$$

$$l_0 == -\sum_{(u_i)\in\mathcal{D}} \sum_{j=1}^{p} t_{i,j} \log(Z_{i,j})$$

$$l_{reg} = \sum_{(u_i,u_j)\in\mathcal{D}} (Z_i^T Z_j - A_{i,j})^2 = vec(Z)^T A\, vec(Z)$$

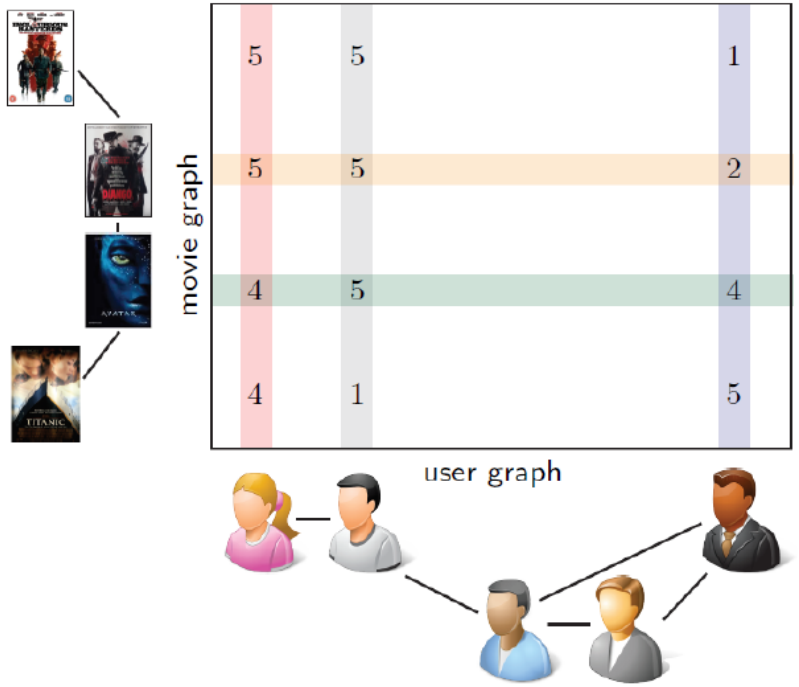
Some applications

- Node classification on citation networks.
  - The input is a citation network where nodes are papers, edges are citation links and optionally bag-of-words features on nodes. The target for each node is a paper category (e.g. stat.ML, cs.LG, ...).

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Matrix completion

- Graph regularization: Matrix completion is the task of filling in the missing entries of a partially observed matrix. A wide range of datasets are naturally organized in matrix form. One example is the movie-ratings matrix, as appears in the Netflix problem: Given a ratings matrix in which each entry (i,j) represents the rating of movie j by user i. When users and movies are organized as graphs then graphs can be used to regularized the matrix completion problem.



Image/molecule classification

- Graph classification: An image is represented as a graph: based on raw pixels (a regular grid and all images have the same graph) or based on superpixels (irregular graph)

Summary on GNN

The key idea is to generate node embeddings based on local neighborhoods.

- Graph convolutional networks
  - Average neighborhood information and stack neural networks.
- GraphSAGE
  - Generalized neighborhood aggregation.
- Gated Graph Neural Networks
  - Neighborhood aggregation + RNNs
- Model wish list :
  - Set W^l of trainable parameters
  - Trainable linear in time in function of $|E|$ or $|V|$.
  - Applicable even if the input graph changes

Next part

Graph kernel

Taxonomy:

- Graph embedding
  - Implicit embedding
    - Graph kernel

Graph kernel

What is a kernel

A kernel, in this context, is a symmetric continuous function that maps
$K : [a, b] \times [a, b] \to \mathbb{R}$
where symmetric means that $K(x, s) = K(s, x)$. K is said to be non-negative definite .

Graph kernel

What is a kernel

The inner product of two vectors a = (a1, a2, ..., an) and b = (b1, b2, ..., bn) is defined as:

$$\langle \mathbf{a} \cdot \mathbf{b} \rangle = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

Kernel Trick

Definition

Graph Kernel Let G be a (finite or infinite) set of graphs. Function $k : G \times G \to \mathbb{R}$ is called a graph kernel if there exists a possibly infinite-dimensional Hilbert space F and a mapping $\phi : G \to f$ such that

- $k(g, g') = \langle \varphi(g), \varphi(g') \rangle$
- $\forall g, g' \in G$ where $\langle ., . \rangle$ denotes a dot product in F.

Kernel Trick

M. Aizerman, E. Braverman, and L. Rozonoer, "Theoretical foundations of the potential function method in pattern recognition learning", Automation and Remote Control, vol. 25, 1964, p. 821-837

Kernel Trick

Definition

Kernel Trick

- Let $\vec{x}$ and $\vec{y}$ be two vectors in $\mathbb{R}^n$
- Let $\varphi(\vec{x})$ and $\varphi(\vec{y})$ be two functions projecting $\vec{x}$ and $\vec{y}$ into $\mathbb{R}^m$.
- with $n \leq m$
- $k(\vec{x}, \vec{y}) = \langle \varphi(\vec{x}), \varphi(\vec{y}) \rangle$
- An explicit representation for φ is not required.
- It suffices to know that $\mathbb{R}^m$ is an inner product space

Kernel Trick

Example:

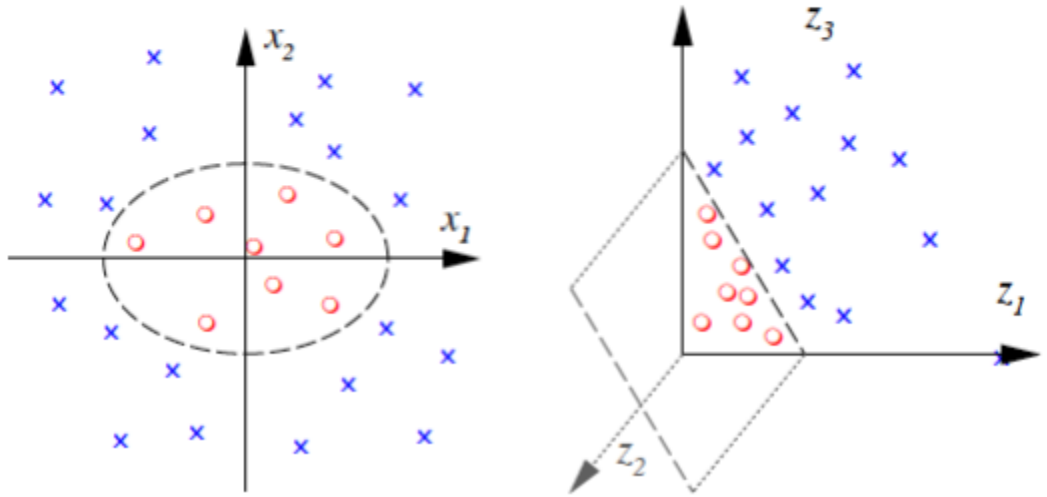
- Let $\vec{x}$ and $\vec{y}$ be two vectors in $\mathbb{R}^2$
- Let $\vec{x} = (x_1, x_2)$ and $\vec{y} = (y_1, y_2)$
- Let $\varphi(\vec{x})$ and $\varphi(\vec{y})$ be two functions projecting $\vec{x}$ and $\vec{y}$ into $\mathbb{R}^3$.
- $k(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle^2$
- $k(\vec{x}, \vec{y}) = x_1^2 y_1^2 + 2x_1 y_1 x_2 y_2 + y_2^2 y_2^2$
- $k(\vec{x}, \vec{y}) = \langle (x_1^2, \sqrt{2} x_1 x_2, x_2^2), (y_1^2, \sqrt{2} y_1 y_2, y_2^2) \rangle$
- $k(\vec{x}, \vec{y}) = \langle \varphi(\vec{x}), \varphi(\vec{y}) \rangle$
- $\varphi(\vec{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$

Kernel Trick

Example:

$$\Phi : R^2 \rightarrow R^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)} x_1 x_2, x_2^2)$$



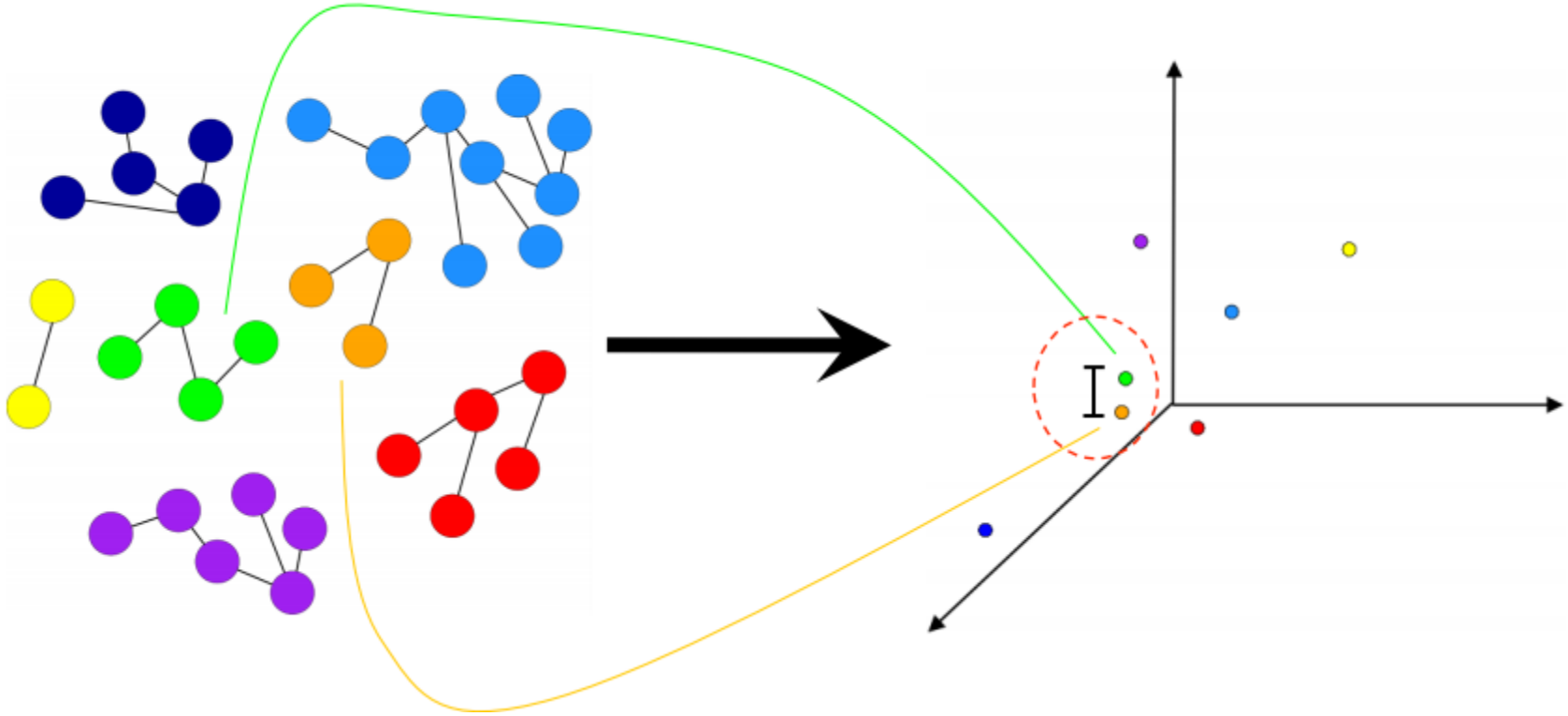
Graph kernel : theoretical issues

- To design a kernel taking the whole graph structure into account amounts to build a complete graph kernel that distinguishes between 2 graphs only if they are not isomorphic
  - Complete graph kernel design is theoretically possible … but practically infeasible (NP-hard)

Graph Kernel Families

- Diffusion kernels (from similarity matrix)
- Convolution kernel
- Walk kernel (from adjacency matrix)

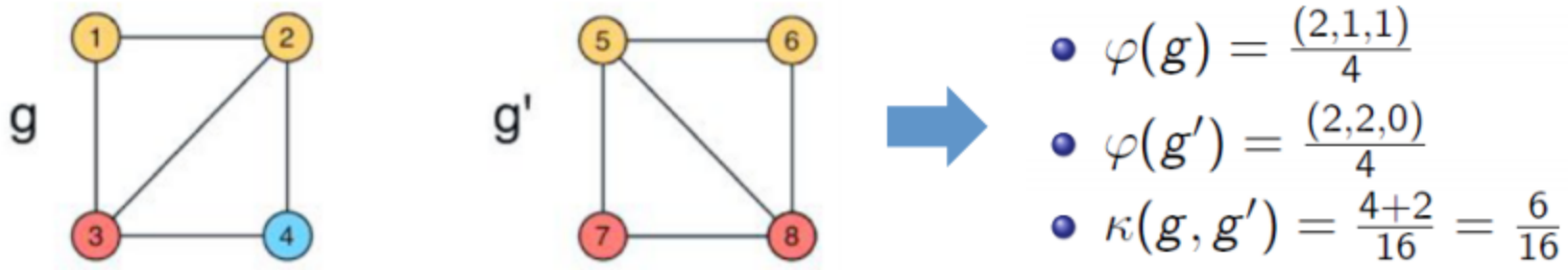
Graph embedding



Very Simple Graph Kernel

Graph Kernel / Graph Embedding [Bunke09]

Example



- $\varphi(g) = \frac{(2,1,1)}{4}$
- $\varphi(g') = \frac{(2,2,0)}{4}$
- $\kappa(g, g') = \frac{4+2}{16} = \frac{6}{16}$

Graph Embedding

$\varphi : G \rightarrow R^n \quad \varphi(g) = (x_1, \ldots, x_n)'$

Many information can be extracted :

→nodes, cliques, paths, walk, …

Random Walks

Principle (Kashima et al., ICML 2003, Gaertner et al., COLT 2003)

- Compare walks in two input graphs G and G'
- Walks are sequences of nodes that allow repetitions of nodes

Elegant computation

- Walks of length k can be computed by looking at the k-th power of the adjacency matrix
- Construct direct product graph of G and G'
- Count walks in this product graph $G_\times=(V_\times,E_\times)$
- Each walk in the product graph corresponds to one walk in G and G'

$$k_\times(G,G') = \sum_{i,j=1}^{|V_\times|} [\sum_{k=0}^{\infty} \lambda^k A_\times^k]_{ij}$$

Graph kernels based on Common Walks

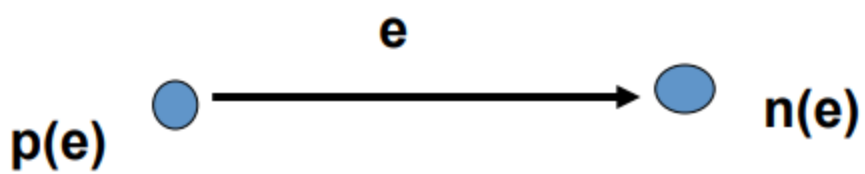
Walk = (possibly infinite) sequence of labels obtained by following edges on the graph

Path = walk with no vertex visited twice

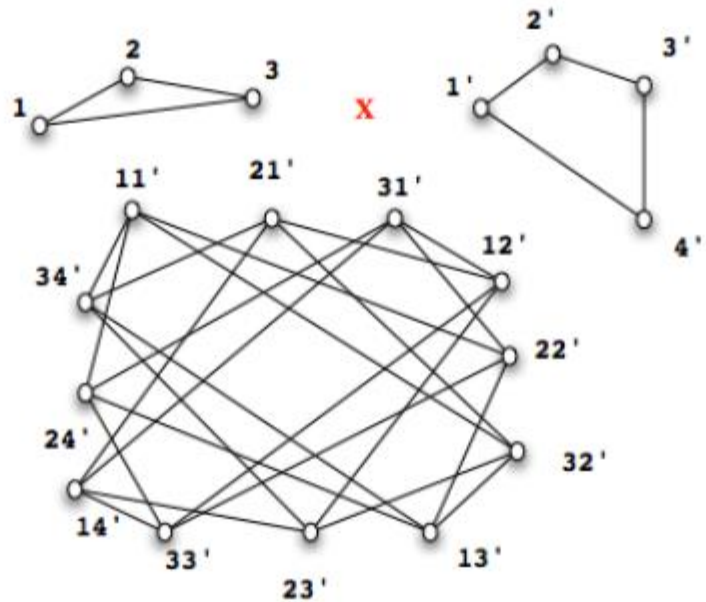
Important concept: direct product of two graphs G1xG2

- V(G1xG2)={(v1,v2), v1 and v2: same labels)
- E(G1xG2)={(e1,e2): e1, e2: same labels, p(e1) and p(e2) same labels, n(e1) and n(e2) same labels}

Same labels : Difficulty to deal with numeric attributes



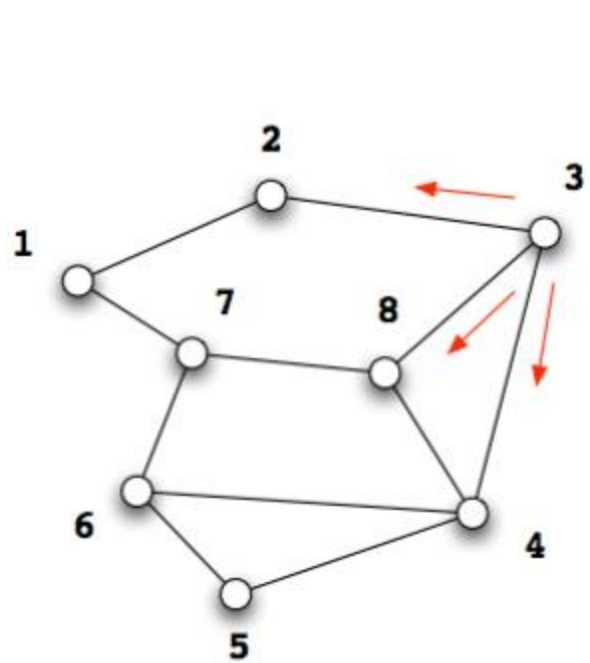
Random Walks:
Explanation : Direct Graph Product



- Formal Definition

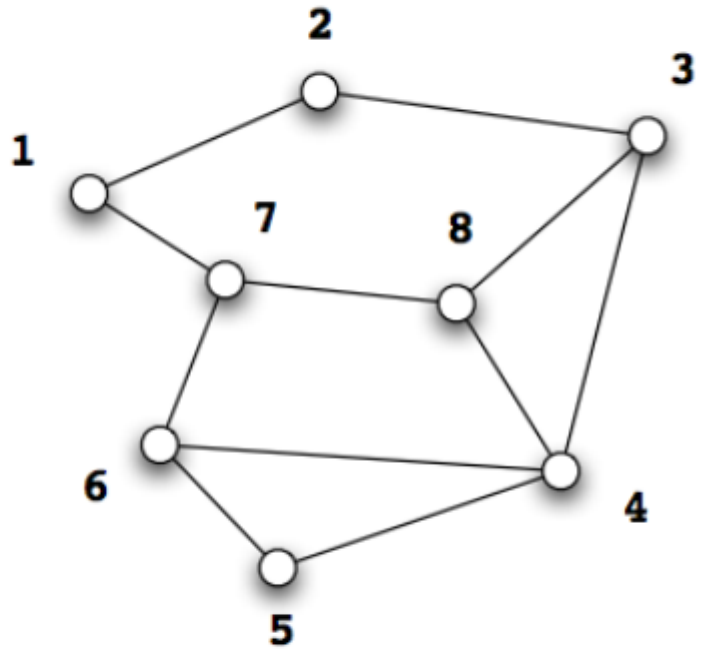
$$V_{\times}(G \times G') = \{(v, v') : v \in V, v' \in V'\}$$
$$E_{\times}(G \times G') = \{((v, v'), (w, w')) : (v, w) \in E, (v', w') \in E'\}$$

Random Walk



- From a vertex i randomly jump to any adjacent vertex j
- Probability of jumping to j proportional to $\tilde{A}_{ij}$

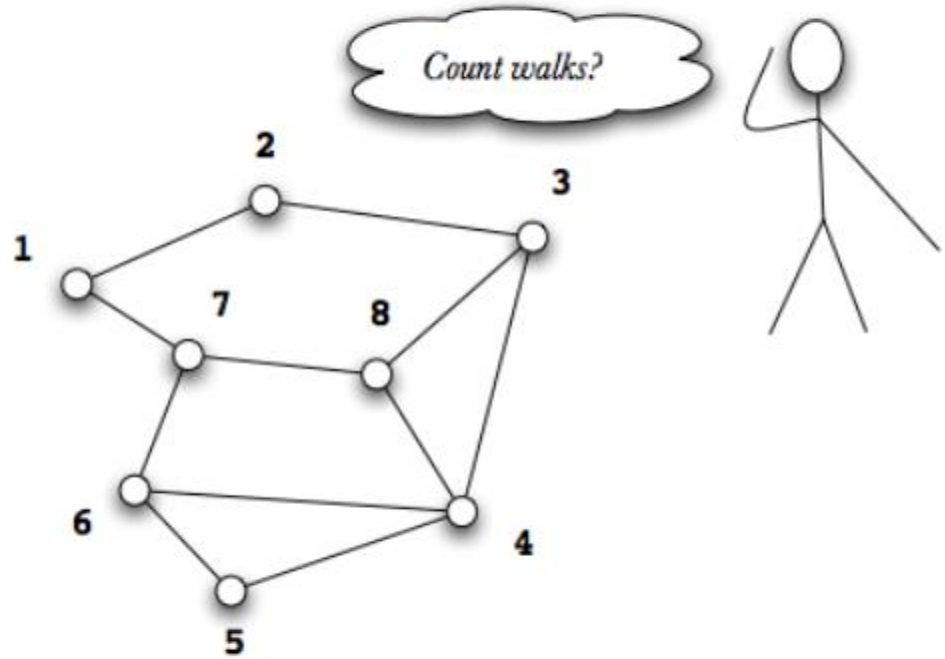
Walks of lengh 2



$$A^2 = \begin{bmatrix} 2 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 3 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 4 & 1 & 1 & 2 & 1 \\ 0 & 0 & 1 & 1 & 2 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 3 & 0 & 2 \\ 0 & 1 & 1 & 2 & 1 & 0 & 3 & 0 \\ 1 & 1 & 1 & 1 & 1 & 2 & 0 & 3 \end{bmatrix}$$

- Entries of A^2 = number of length 2 walks
- Entries of $\tilde{A}^2$ = probability of length 2 walks

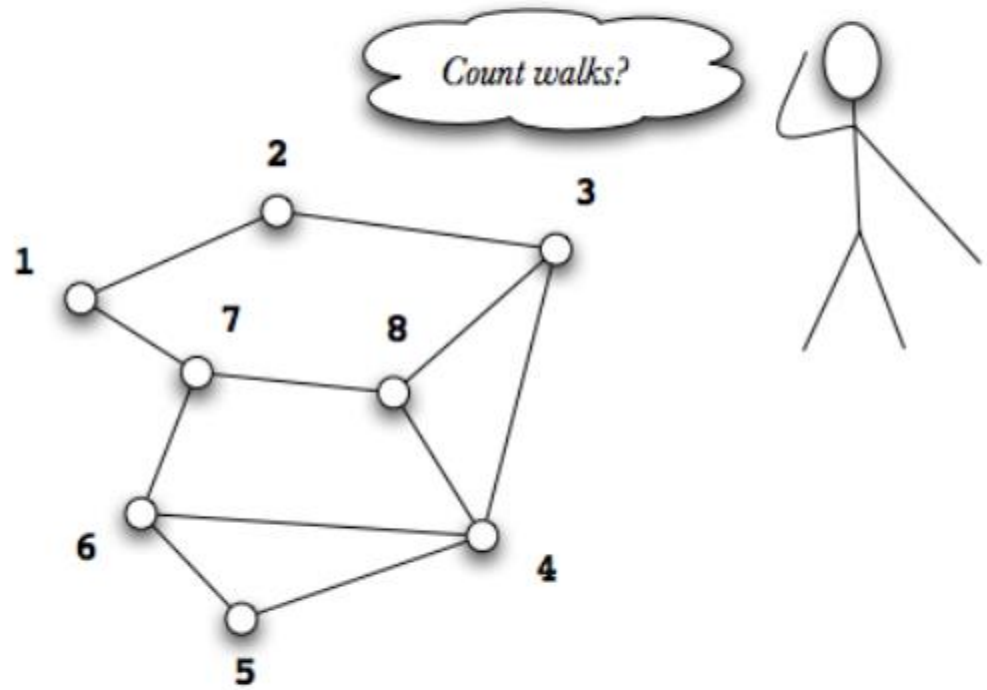
Random Walks: Explanation : Idea



- Count number of walks between two nodes
- Two nodes are similar if they are connected by many walks

- Does not work :(
- If graph has cycles then number of walks goes to ∞

Random Walks: Explanation : a better idea



- Discount contribution of longer walks
- Count number of walks between two nodes
- Two nodes are similar if they are connected by many walks

- Works if discounting factor chosen appropriately!

Random Walks:
Explanation : Diffusion Kernels

Discounting Factor:

- Discount a k length walk by $\lambda^k/k!$ for $0 \leq \lambda \leq 1$

Similarity:

- Similarity defined as

$$k(i,j) = \left[\sum_k \frac{\lambda^k}{k!} A^k\right]_{ij} = [\exp(\lambda A)]_{ij}$$

Random Walks:
Explanation : Graph Comparison

Random Walk on Product Graph:

- Equivalent to simultaneous random walk on input graphs

Kernel Definition:

$$k(G, G') = \frac{1}{|G|\,|G'|} \sum_k \frac{\lambda^k}{k!}\, \mathbf{e}^\top A_\times^k\, \mathbf{e} = \frac{1}{|G|\,|G'|}\, \mathbf{e}^\top \exp(\lambda A_\times)\, \mathbf{e}$$

Efficient computation ?

Product Graph is Huge:

- If G and G' have n vertices then product graph has n^2 vertices
- Adjacency matrix $A_\times$ is of size $n^2 \times n^2$

$$\underbrace{\exp(\lambda A_\times)}_{O(n^6)\,!}.$$

Summary on graph kernels

- Comparing paths of two different graphs is polynomial
- Comparing subgraphs of two different graphs optimally is not guaranteed in polynomial time.
- Tradeoff between expressivity of the kernel and the computation time.
- Kernels have been extended to take into account numeric attributes.

Some experiments

- Comparison between graph kernels and graph neural networks.
- Taken from :
  - **How Powerful are Graph Neural Networks? (2018)**
  - Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka

Social networks datasets. IMDB-BINARY and IMDB-MULTI are movie collaboration datasets. Each graph corresponds to an ego-network for each actor/actress, where nodes correspond to actors/actresses and an edge is drawn betwen two actors/actresses if they appear in the same movie. Each graph is derived from a pre-specified genre of movies, and the task is to classify the genre graph it is derived from. REDDIT-BINARY and REDDIT-MULTI5K are balanced datasets where each graph corresponds to an online discussion thread and nodes correspond to users. An edge was drawn between two nodes if at least one of them responded to another's comment. The task is to classify each graph to a community or a subreddit it belongs to. COLLAB is a scientific collaboration dataset, derived from 3 public collaboration datasets, namely, High Energy Physics, Condensed Matter Physics and Astro Physics. Each graph corresponds to an ego-network of different researchers from each field. The task is to classify each graph to a field the corresponding researcher belongs to.

Bioinformatics datasets. MUTAG is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds with 7 discrete labels. PROTEINS is a dataset where nodes are secondary structure elements (SSEs) and there is an edge between two nodes if they are neighbors in the amino-acid sequence or in 3D space. It has 3 discrete labels, representing helix, sheet or turn. PTC is a dataset of 344 chemical compounds that reports the carcinogenicity for male and female rats and it has 19 discrete labels. NCI1 is a dataset made publicly available by the National Cancer Institute (NCI) and is a subset of balanced datasets of chemical compounds screened for ability to suppress or inhibit the growth of a panel of human tumor cell lines, having 37 discrete labels.

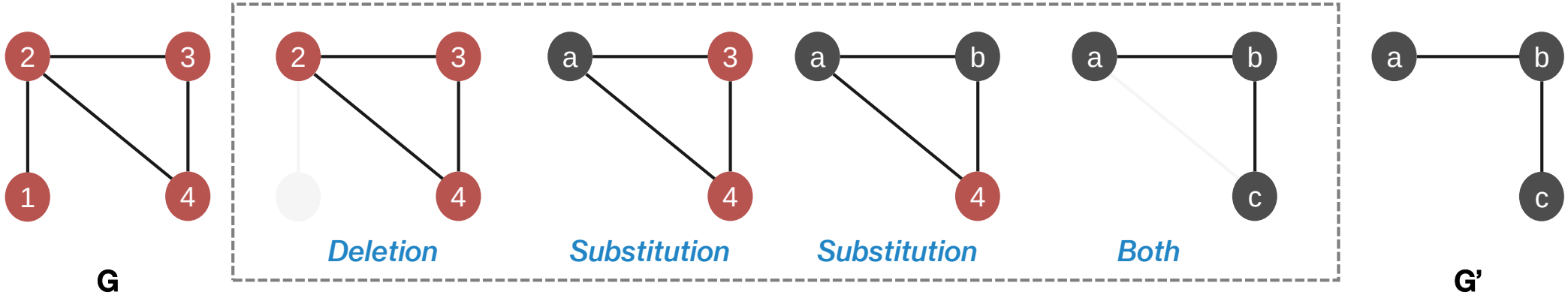
	Datasets	IMDB-B	IMDB-M	RDT-B	RDT-M5K	COLLAB	MUTAG	PROTEINS	PTC	NCI1
Datasets	# graphs	1000	1500	2000	5000	5000	188	1113	344	4110
	# classes	2	3	2	5	3	2	2	2	2
	Avg # nodes	19.8	13.0	429.6	508.5	74.5	17.9	39.1	25.5	29.8
Baselines	WL subtree	73.8 ± 3.9	50.9 ± 3.8	81.0 ± 3.1	52.5 ± 2.1	78.9 ± 1.9	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	**86.0 ± 1.8** *
	DCNN	49.1	33.5	–	–	52.1	67.0	61.3	56.6	62.6
	PATCHYSAN	71.0 ± 2.2	45.2 ± 2.8	86.3 ± 1.6	49.1 ± 0.7	72.6 ± 2.2	**92.6 ± 4.2** *	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9
	DGCNN	70.0	47.8	–	–	73.7	85.8	75.5	58.6	74.4
	AWL	74.5 ± 5.9	51.5 ± 3.6	87.9 ± 2.5	54.7 ± 2.9	73.9 ± 1.9	87.9 ± 9.8	–	–	–
GNN variants	GIN-ϵ (SUM–MLP)	**74.3 ± 5.1**	**52.1 ± 3.6**	**92.2 ± 2.3**	**57.0 ± 1.7**	**80.1 ± 1.9**	**89.0 ± 6.0**	**75.9 ± 3.8**	63.7 ± 8.2	**82.7 ± 1.6**
	GIN-0 (SUM–MLP)	**75.1 ± 5.1**	**52.3 ± 2.8**	**92.4 ± 2.5**	**57.5 ± 1.5**	**80.2 ± 1.9**	**89.4 ± 5.6**	**76.2 ± 2.8**	**64.6 ± 7.0**	**82.7 ± 1.7**
	SUM–1-LAYER	74.1 ± 5.0	**52.2 ± 2.4**	90.0 ± 2.7	55.1 ± 1.6	**80.6 ± 1.9**	**90.0 ± 8.8**	**76.2 ± 2.6**	63.1 ± 5.7	82.0 ± 1.5
	MEAN–MLP	73.7 ± 3.7	**52.3 ± 3.1**	50.0 ± 0.0 † (71.2 ± 4.6)	20.0 ± 0.0 † (41.3 ± 2.1)	79.2 ± 2.3	83.5 ± 6.3	75.5 ± 3.4	**66.6 ± 6.9**	80.9 ± 1.8
	MEAN–1-LAYER	74.0 ± 3.4	51.9 ± 3.8	50.0 ± 0.0 † (69.7 ± 3.2)	20.0 ± 0.0 † (39.7 ± 2.4)	79.0 ± 1.8	85.6 ± 5.8	76.0 ± 3.2	64.2 ± 4.3	80.2 ± 2.0
	MAX–MLP	73.2 ± 5.8	51.1 ± 3.6	–	–	–	84.0 ± 6.1	76.0 ± 3.2	64.6 ± 10.2	77.8 ± 1.3
	MAX–1-LAYER	72.3 ± 5.3	50.9 ± 2.2	–	–	–	85.1 ± 7.6	75.9 ± 3.2	63.9 ± 7.7	77.7 ± 1.5

Table 1: Classification accuracies (%). † indicate test accuracies (equal to chance rates) when all nodes have the same feature vector. We also report in the parentheses the test accuracies when the node degrees are used as input node features. The best-performing GNNs are highlighted with boldface. On datasets where GINs' accuracy is not strictly the highest among GNN variants, GINs are comparable to the best because paired t-test at significance level 10% does not distinguish GINs from the best. If a baseline performs better than all GNNs, we highlight it with boldface and asterisk.

Distance onto graph space

- Graph comparison is not a trivial task
  - Due to the wide range of patterns (i.e : comparing the number of nodes is not enough).
- Closely related to graph matching problems

Graph distance : Intuition



Vertices operations

- OP_1: 1 → ϵ ⇒ C_1
- OP_3: 2 → a ⇒ C_2
- OP_4: 3 → b ⇒ C_3
- OP_6: 4 → c ⇒ C_4

Edges operations

- OP_2: (1,2) → ϵ ⇒ C_5
- OP_5: (2,3) → (a,b) ⇒ C_6
- OP_7: (3,4) → (b,c) ⇒ C_7
- OP_8: (2,4) → ϵ ⇒ C_8

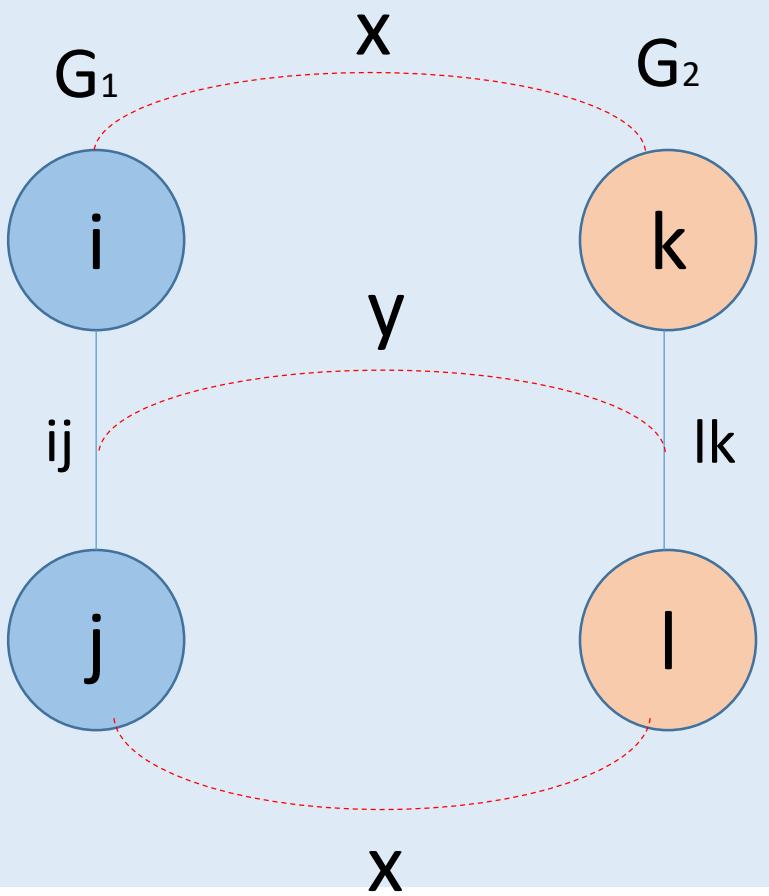
This solution costs:

$$d = \sum_{op_i \in operations} C(op_i)$$

Graph Distance : Modelling

- Notations

Edit operation	Variable	Cost
Substitution of vertex i by vertex k	$x_{i,k}$	$c_{i,k}$
Deletion of vertex i	u_i	$c_{i,\epsilon}$
Insertion of vetex k	v_k	$c_{\epsilon,k}$
Substitution of edge ij by edge kl	$y_{ij,kl}$	$c_{ij,kl}$
Deletion of edge ij	e_{ij}	$c_{ij,\epsilon}$
Insertion of edge kl	f_{kl}	$c_{\epsilon,kl}$



Graph distance : Integer Linear Program

- Objective function:

Vertex substitutions | Edge substitutions | vertex deletions

$$C(\mathbf{x},\mathbf{y},\mathbf{u},\mathbf{v},\mathbf{e},\mathbf{f}) = \sum_{i\in V_1}\sum_{k\in V_2} c_{i,k}\cdot x_{i,k} + \sum_{ij\in E_1}\sum_{kl\in E_2} c_{ij,kl}\cdot y_{ij,kl} + \sum_{i\in V_1} c_{i,\epsilon}\cdot u_i + \sum_{k\in V_2} c_{\epsilon,k}\cdot v_k + \sum_{ij\in E_1} c_{ij,\epsilon}\cdot e_{ij} + \sum_{kl\in E_2} c_{\epsilon,kl}\cdot f_{kl} \quad (1)$$

vertex insertions | edge deletions | edge insertions

Graph distance : Integer Linear Program

Vertices mapping constraints

deletion substitutions

$$u_i + \sum_{k \in V_2} x_{i,k} = 1 \quad \forall i \in V_1$$

Insertion

$$v_k + \sum_{i \in V_1} x_{i,k} = 1 \quad \forall k \in V_2$$

Edges mapping constraints

deletion substitutions

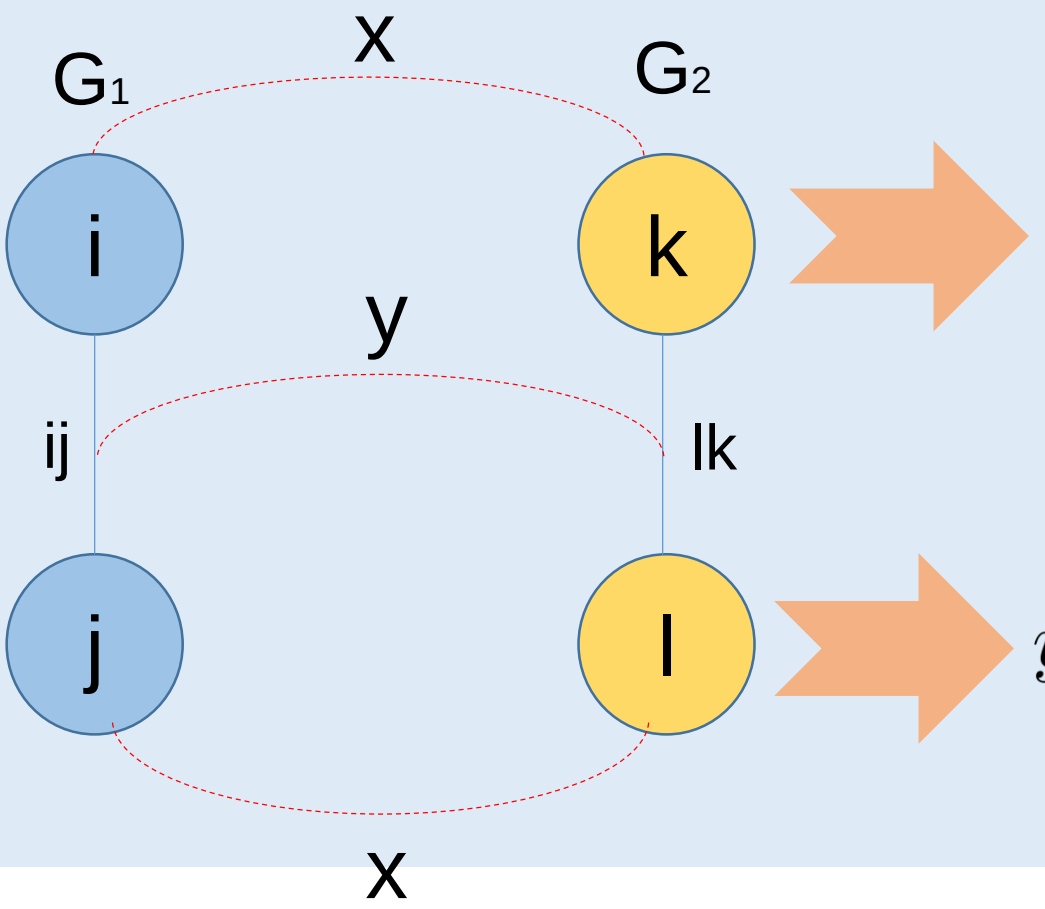
$$e_{ij} + \sum_{kl \in E_2} y_{ij,kl} = 1 \quad \forall ij \in E_1$$

$$f_{kl} + \sum_{ij \in E_1} y_{ij,kl} = 1 \quad \forall kl \in E_2$$

Insertion

Graph distance : Integer Linear Program

- Topological constraints



$$y_{ij,kl} \leq x_{i,k} \quad \forall (ij,kl) \in E_1 \times E_2$$

$$y_{ij,kl} \leq x_{j,l} \quad \forall (ij,kl) \in E_1 \times E_2$$

Distance onto graph space

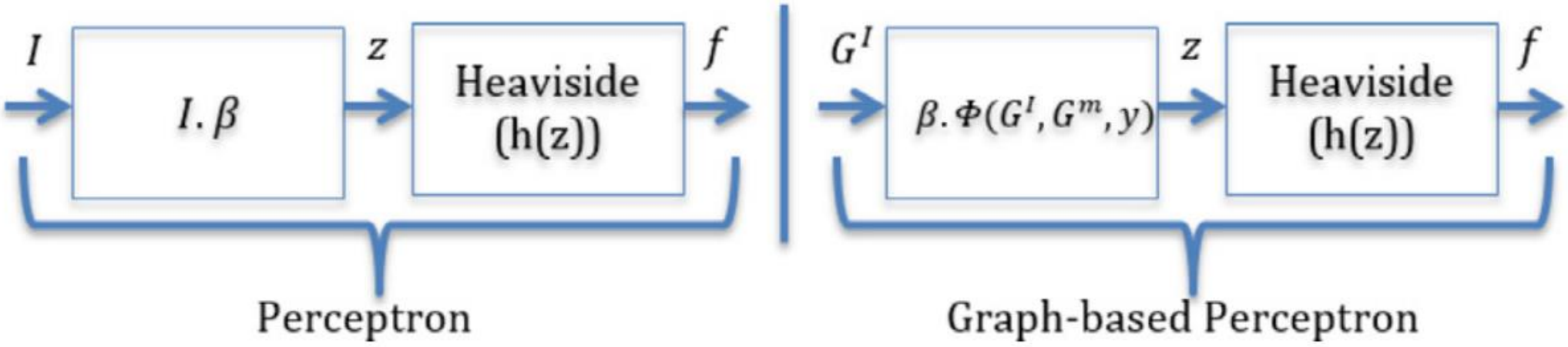
- NP-Hard problem
  - Finding the optimal solution is not guaranteed in polynomial time.
- Models : ILP models
- Solvers : Exact and heuristic methods (Matheuristic)

Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, Sébastien Adam: **New binary linear programming formulation to compute the graph edit distance.** Pattern Recognition 72: 254-265 (2017)

End-to-end learning techniques onto graph space

- Neural networks
  - « Usually » deal with Euclidean data (vectors, matrices, tensors, …).
  - deal with sequence -> RNN
- Wanted : Neural networks dealing with graphs
  - Input : a graph
  - Output : a graph (and a scalar)

Graph Neural Network onto graph space



Key points:

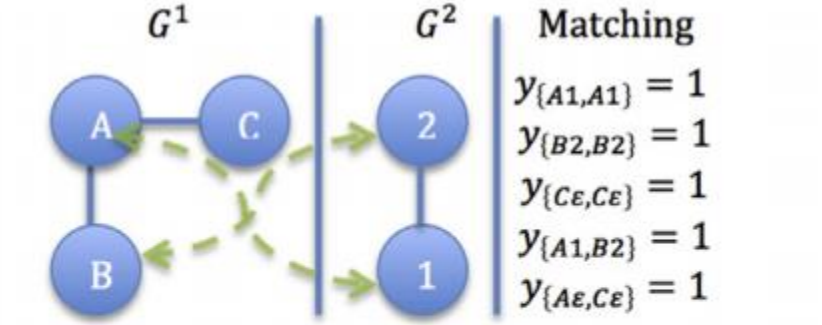
1. Each neuron is a graph
2. ϕ is a parametrized version the graph distance

Parametrized graph distance

- Each neuron is a graph
- The input is a graph
- We have 2 graphs :
  - Graph distance computation
- Graph distance should be trainable

$$d(G^1, G^2, y, \beta) = \beta \Phi^T (G^1, G^2, y)$$

$$= \cdots + \beta_a \cdot d_V (L^1_{\pi(a)}, L^2_a) + \\ + \beta_{ab} \cdot d_E (L^1_{\pi(a)\pi(b)}, L^2_{ab}) + \cdots$$



$$d(G^1, G^2, y, \beta) = \\ y_{\{A1,A1\}} d(A,1)\beta_1 + d(B,2)\beta_2 + y_{\{C\varepsilon,C\varepsilon\}} d(C,\varepsilon)\beta_{\{\varepsilon\}} + \\ y_{\{A1,B2\}} d(AB,12)\beta_{\{12\}} + y_{\{A\varepsilon,C\varepsilon\}} d(AC,\varepsilon\varepsilon)\beta_{\{\varepsilon\varepsilon\}}$$

Learning the graph neural network

- One loss :

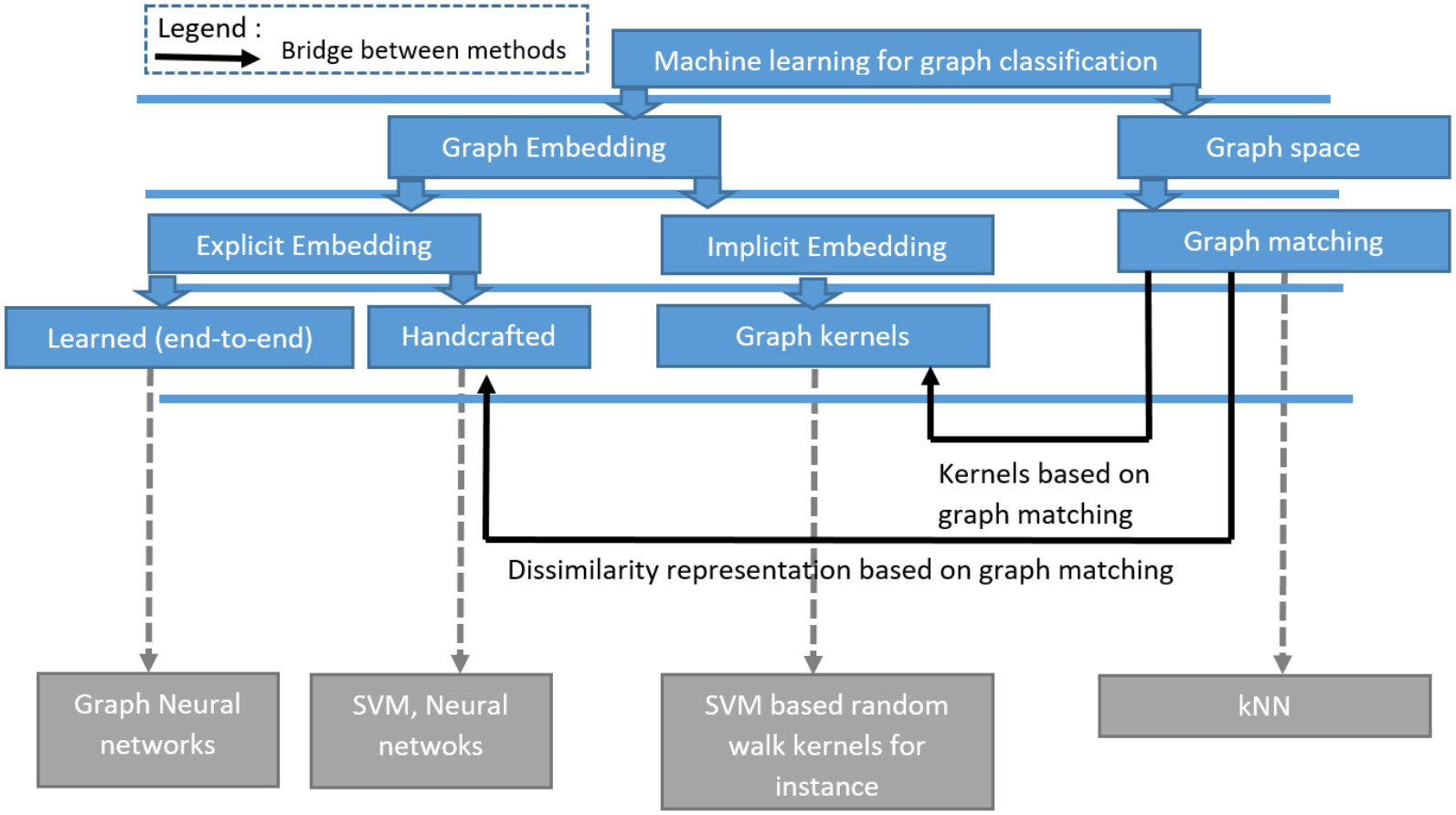
$$\min_{\beta,y} \sum_{(G^k,c_k)\in TrS} l(G^k, c_k, G^m, y_k, \beta)$$

$$l = \frac{1}{2}\left(c_k - heaviside(\beta \cdot \Phi(G^k, G^m, y_k^*))\right)^2$$

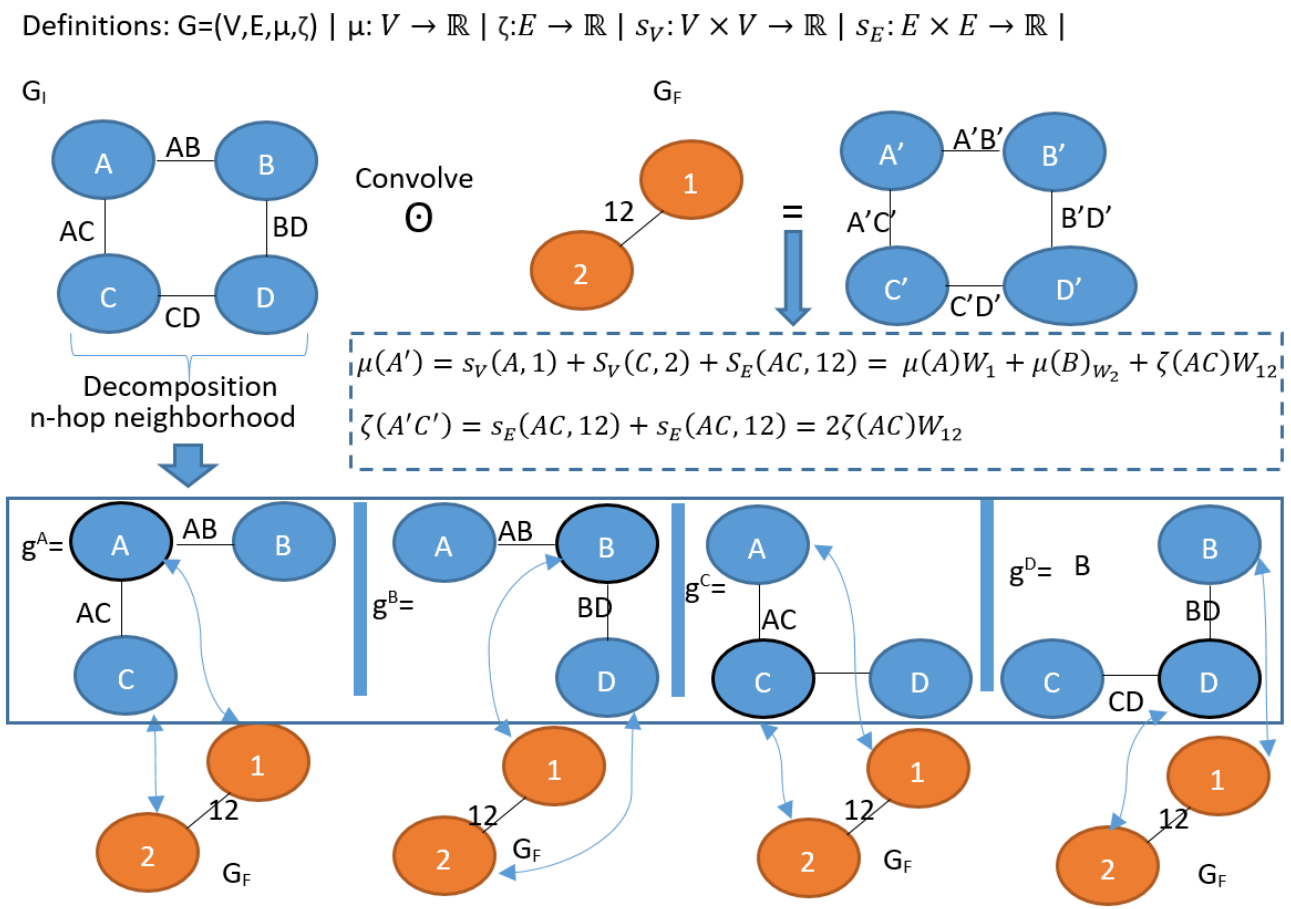
- Minimizer :
  - Gradient descent for the weights (β)
  - Graph distance solver for the *y* variables

Maxime Martineau, Romain Raveaux, Donatello Conte, Gilles Venturini, Learning error-correcting graph matching with a multiclass neural network, Pattern Recognition Letters, 2018

Graph convolution neural network onto graph space



Graph convolution neural network onto graph space



Thank you

- Any questions ?