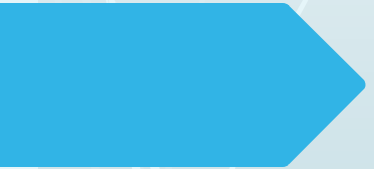


Apprentissage avec des graphes – Partie 1

IFT 6758A / IFT3700

Automne 2024

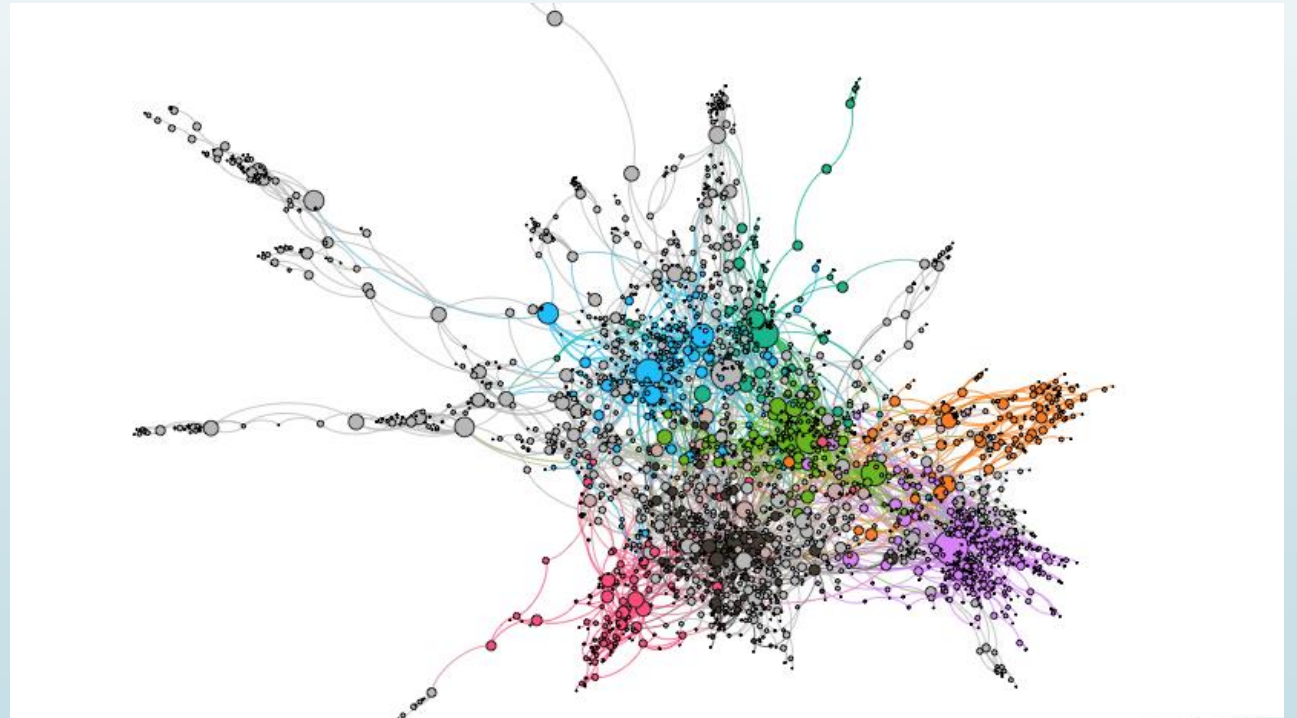
Gauthier Gidel



Omniprésence de graphes

► Les graphes sont des structures générales pour décrire des systèmes complexes

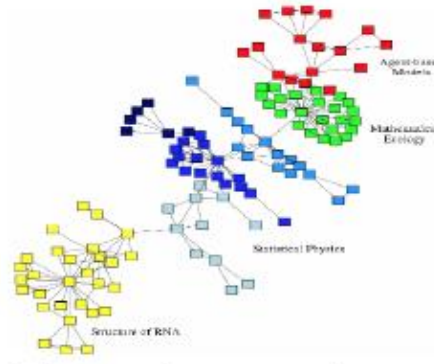
► $G = (V, E)$



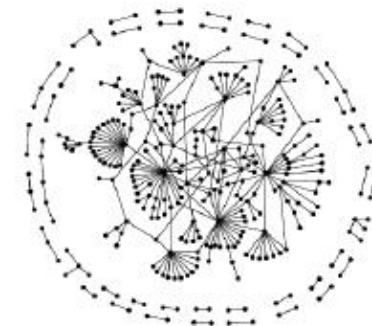
Omniprésence de graphes



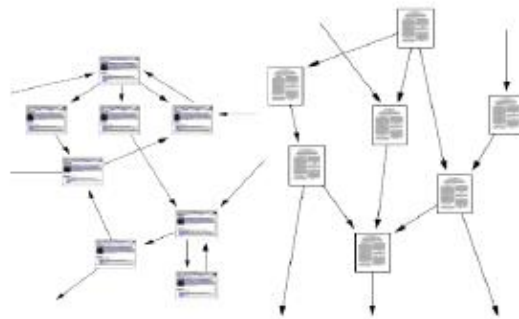
Social networks



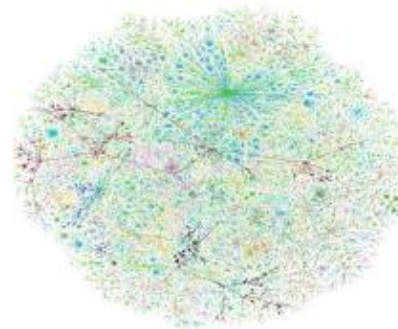
Economic networks



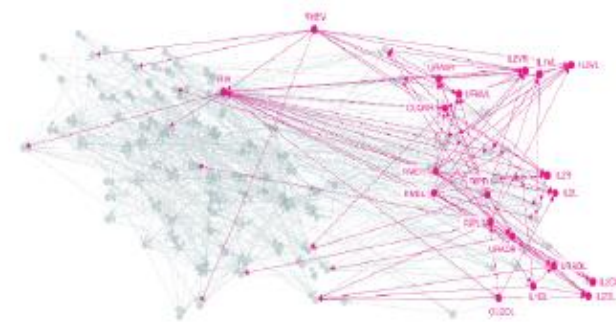
Biomedical networks



Information networks:
Web & citations



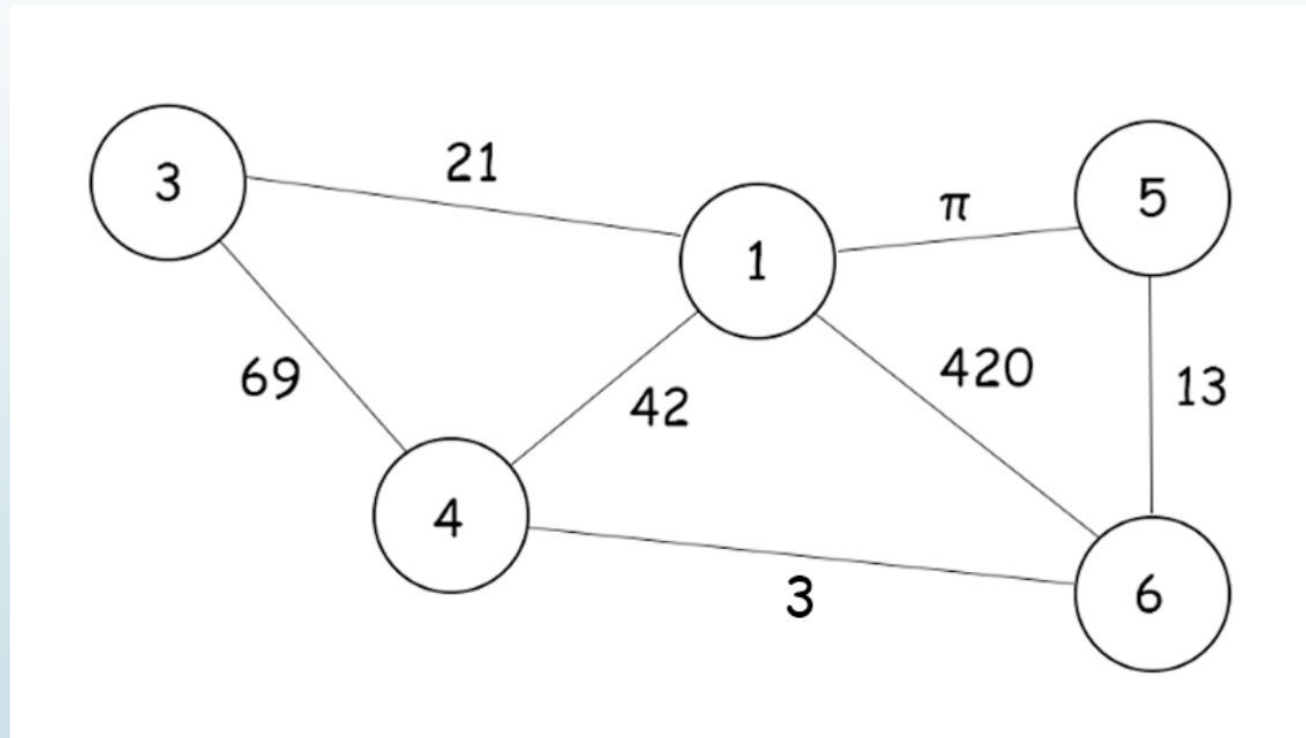
Internet



Networks of neurons

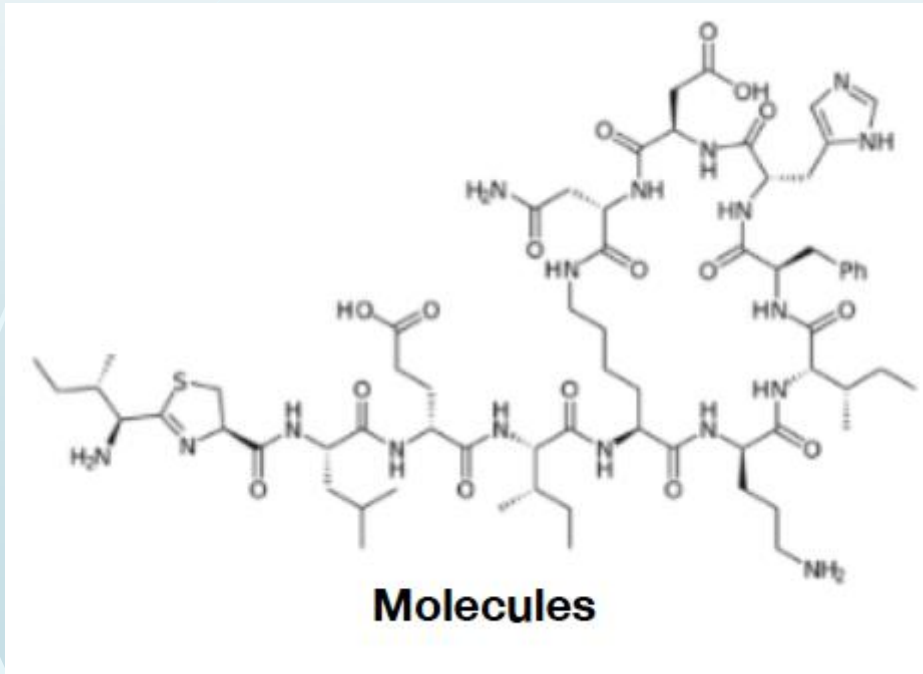
Qu'est-ce qu'un graphe ?

- Les graphes peuvent avoir **des étiquettes** sur leurs **arêtes et/ou leurs nœuds**

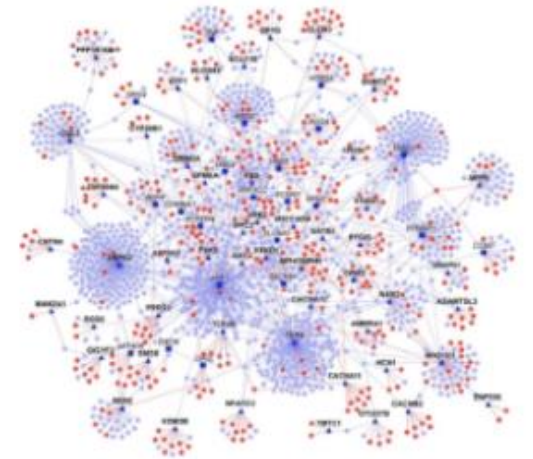


- Les étiquettes peuvent également être considérées comme des poids

Qu'est-ce qu'un graphe ?



Protein interaction networks

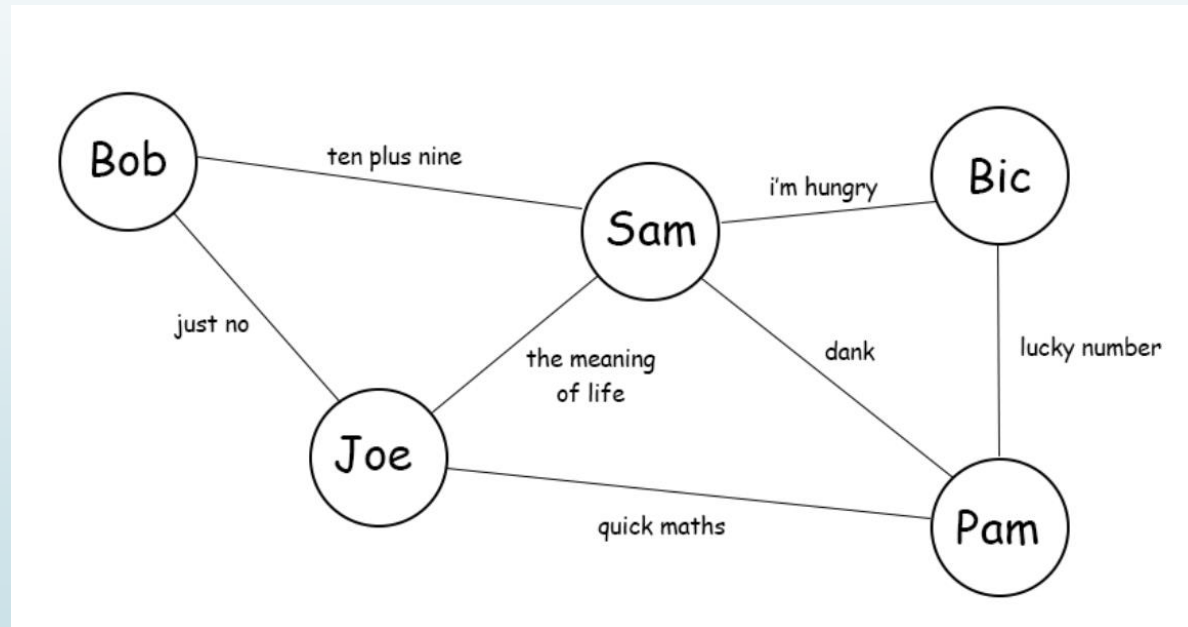


Road maps



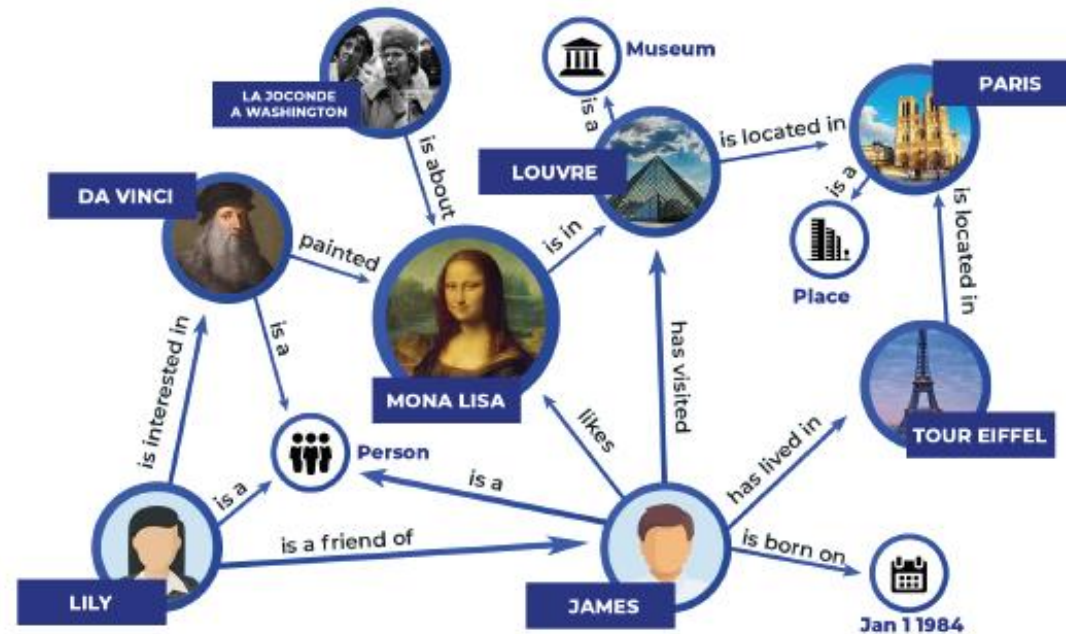
Qu'est-ce qu'un graphe ?

- Les étiquettes n'ont pas besoin d'être numériques, elles peuvent être **textuelles**.



- Les étiquettes n'ont pas besoin d'être uniques; Il est possible et parfois même utile de donner la même étiquette à plusieurs noeuds.

Qu'est-ce qu'un graphe ?



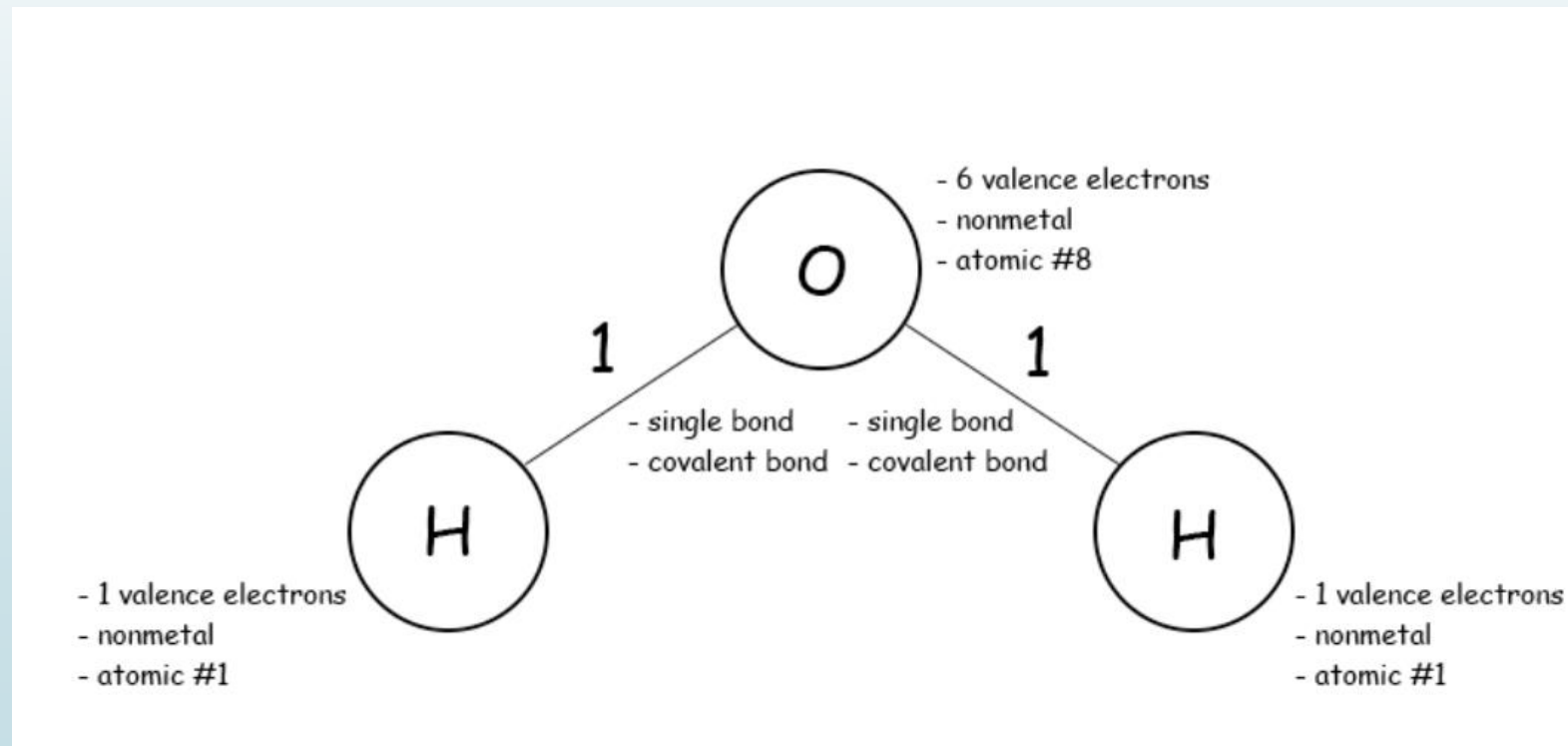
Wikipedia
Google Knowledge graph

....



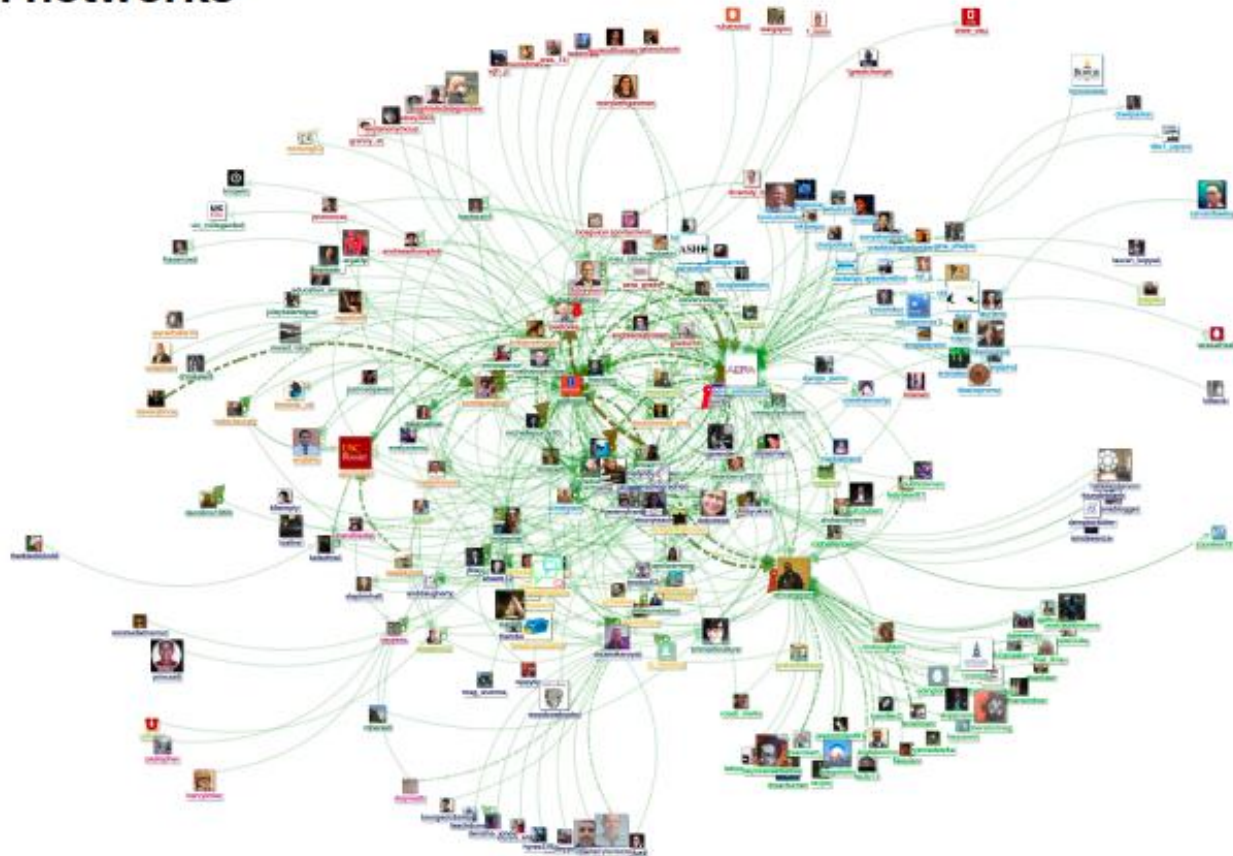
Qu'est-ce qu'un graphe ?

- Les graphes peuvent avoir des caractéristiques (attributs).

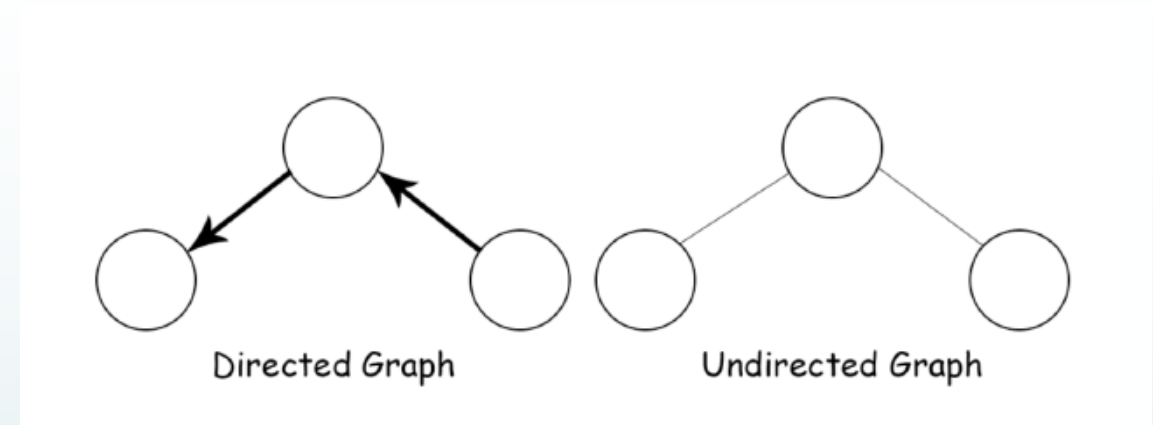


Qu'est-ce qu'un graphe ?

Social networks



Qu'est-ce qu'un graphe ?



Catégories de graphes :

- **Hétérogène** : composé de différents types de nœuds
- **Homogène** : composé du même type de nœuds

Et

- **Statique** : les nœuds et les arêtes ne changent pas, rien n'est ajouté ou enlevé
- **Dynamique** : les nœuds et les arêtes changent, ajoutent, suppriment, déplacent, etc.

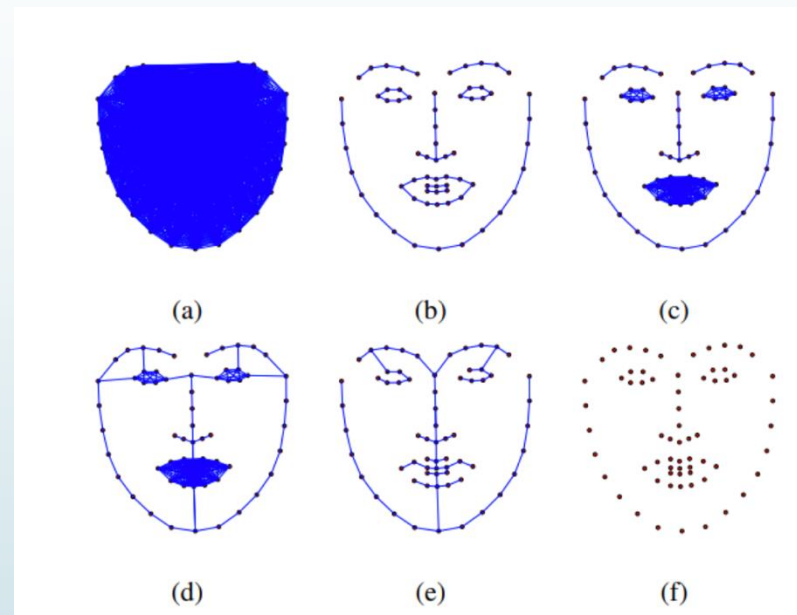


Pourquoi les graphes sont-ils utiles ?

- Il s'agit d'une structure de données **très flexible** qui généralise de nombreuses autres structures de données.
 - Par exemple, s'il n'y a pas d'arêtes, **alors cela devient un ensemble**;
 - S'il n'y a que des arêtes « verticales » et que deux nœuds quelconques sont reliés par exactement un chemin, alors nous avons un **arbre**.
- Les **nœuds et les arêtes** proviennent généralement d'une connaissance ou d'une **intuition d'expert sur le problème**.
 - p. ex., atomes dans les molécules, utilisateurs dans un réseau social, villes dans un système de transport, joueurs dans le sport d'équipe, neurones dans le cerveau, objets en interaction dans un système physique dynamique, et pixels, cadres de délimitation ou masques de segmentation dans les images

Pourquoi les graphes sont-ils utiles ?

- La plupart des problèmes de ML/CV peuvent être visualisés sous forme de graphes
- [from Antonakos et al., CVPR, 2015](#)



- Les graphes donnent beaucoup de flexibilité et peuvent donner une perspective très différente et intéressante sur un problème
- Certaines modalités sont intrinsèquement des graphes (réseaux sociaux)

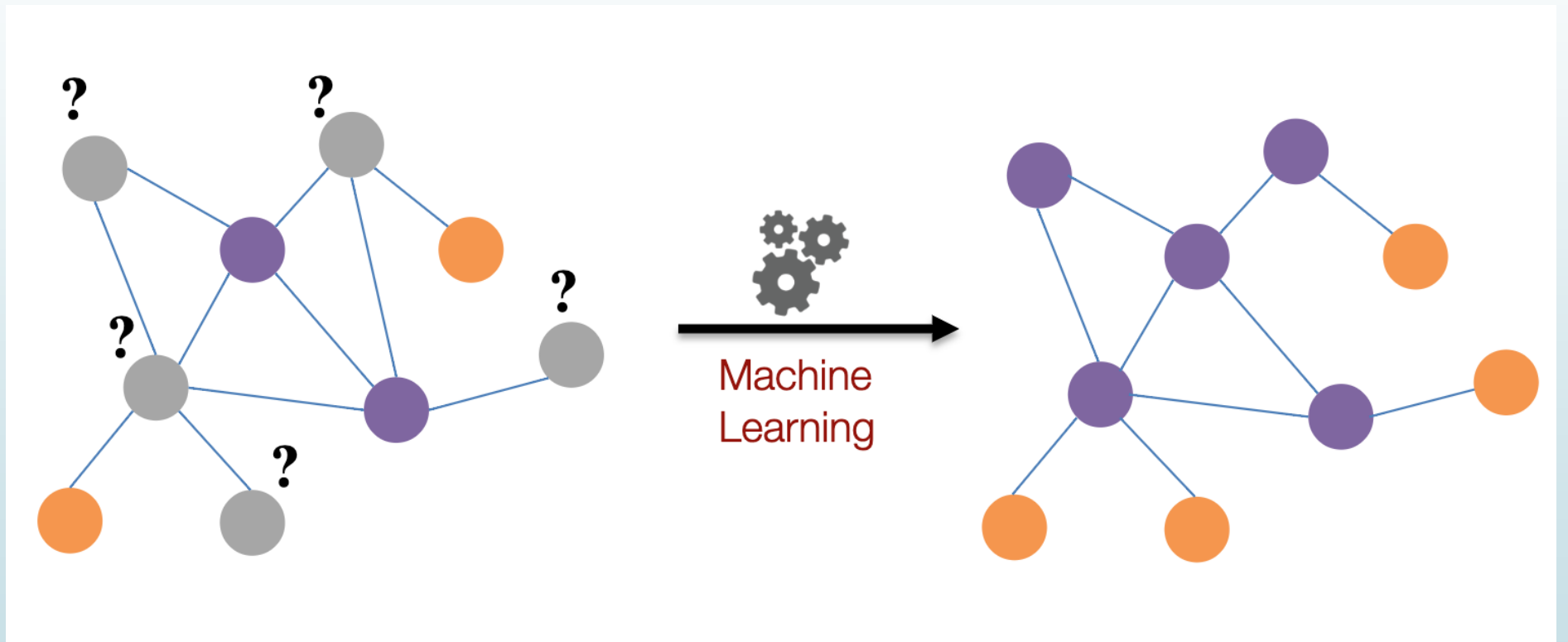


Tâches d'exploration de graphes

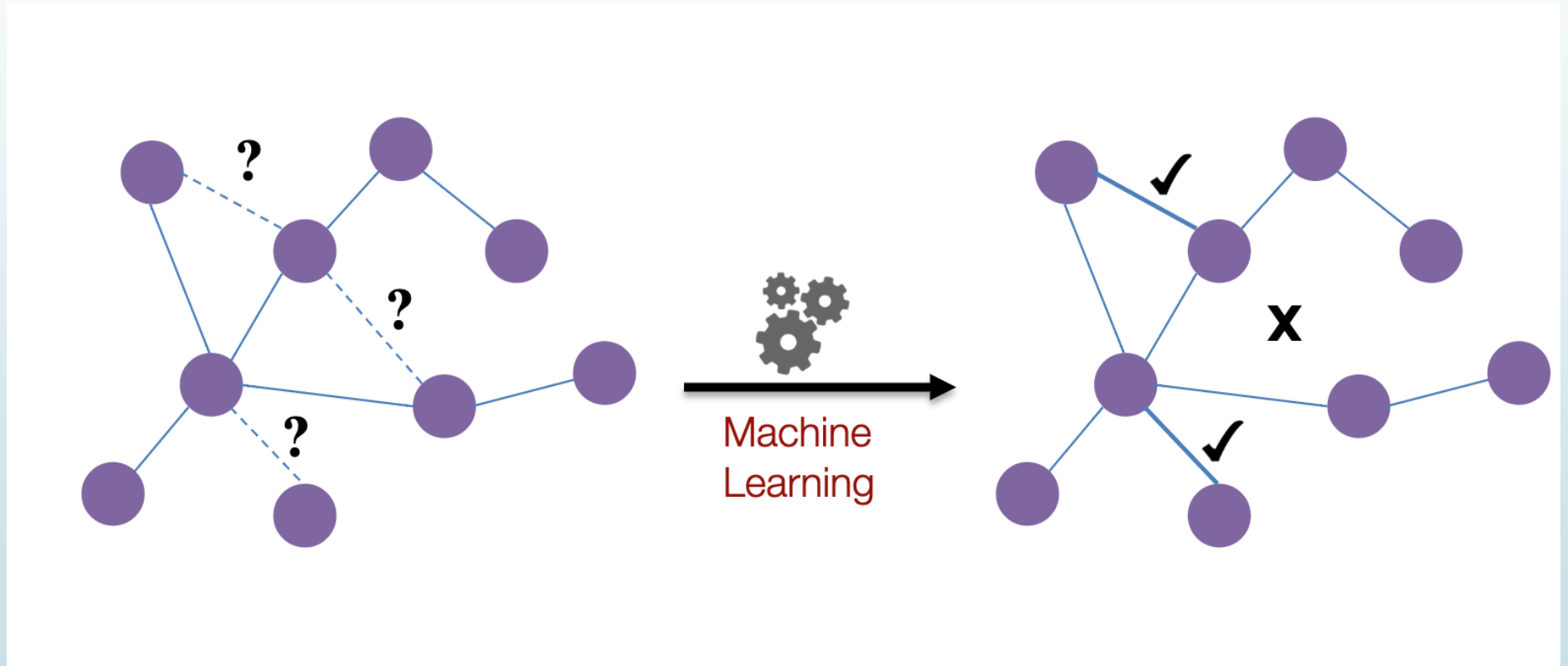
Tâches ML classiques dans les graphes:

1. Classification des nœuds : prédire un type de nœud donné
2. Prédiction de liaison : prédire si deux nœuds sont liés
3. Détection communautaire : identifier les grappes de nœuds densément liées
4. Similitude de réseau : à quel point deux (sous-)réseaux sont-ils similaires

Classification des nœuds

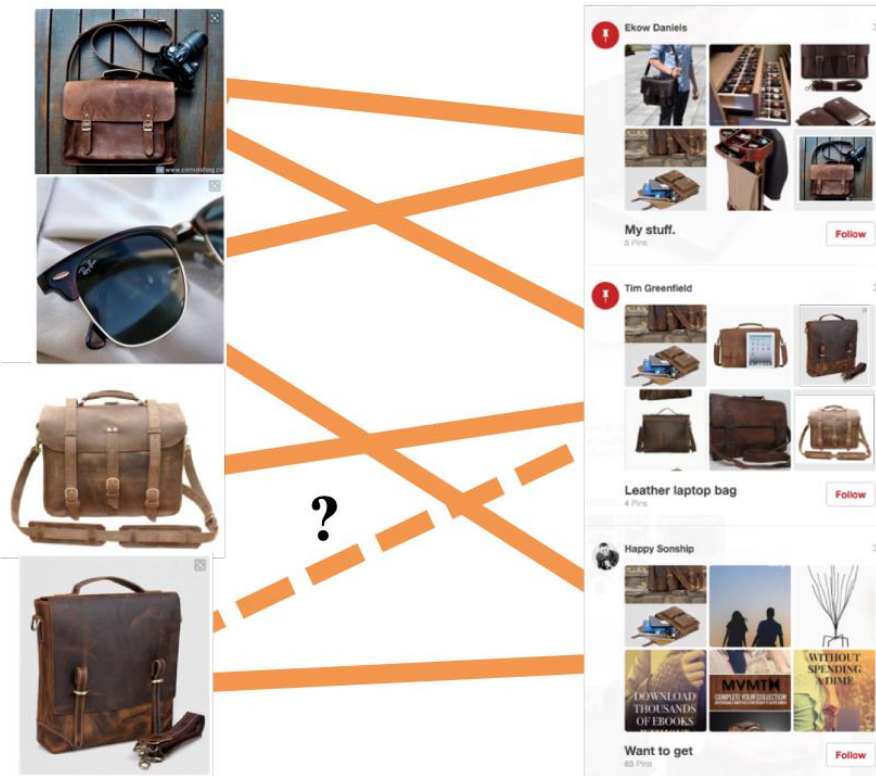


Prédiction de lien



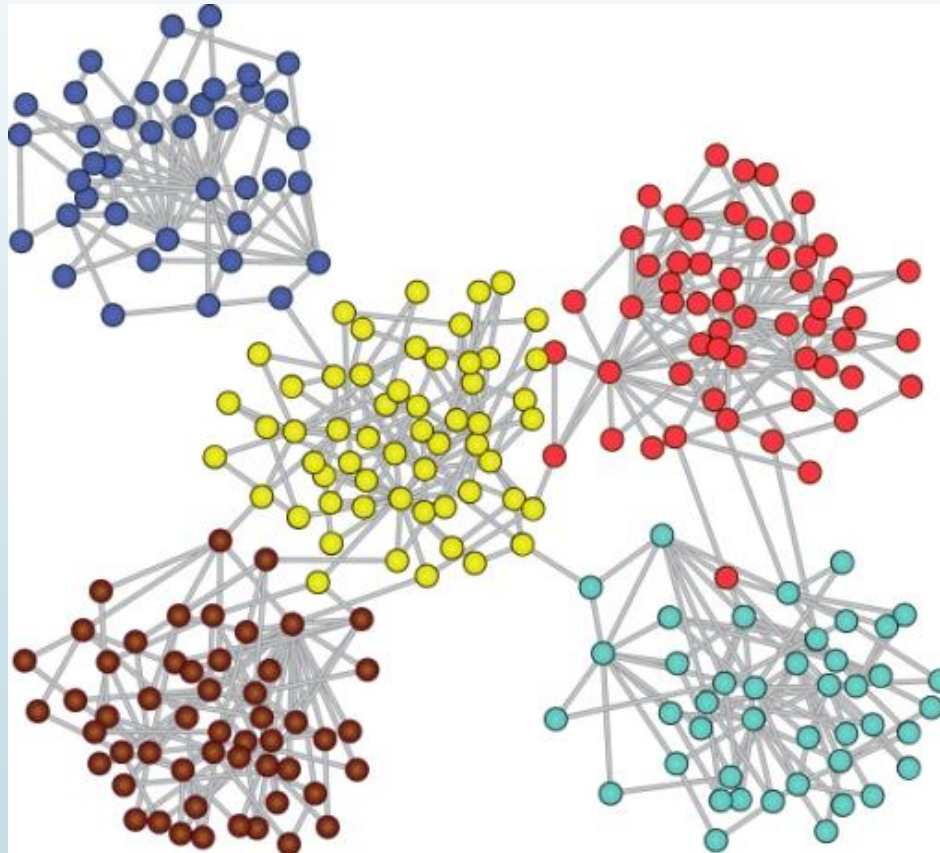
Prédiction de liens

Content
recommendation is
link prediction!

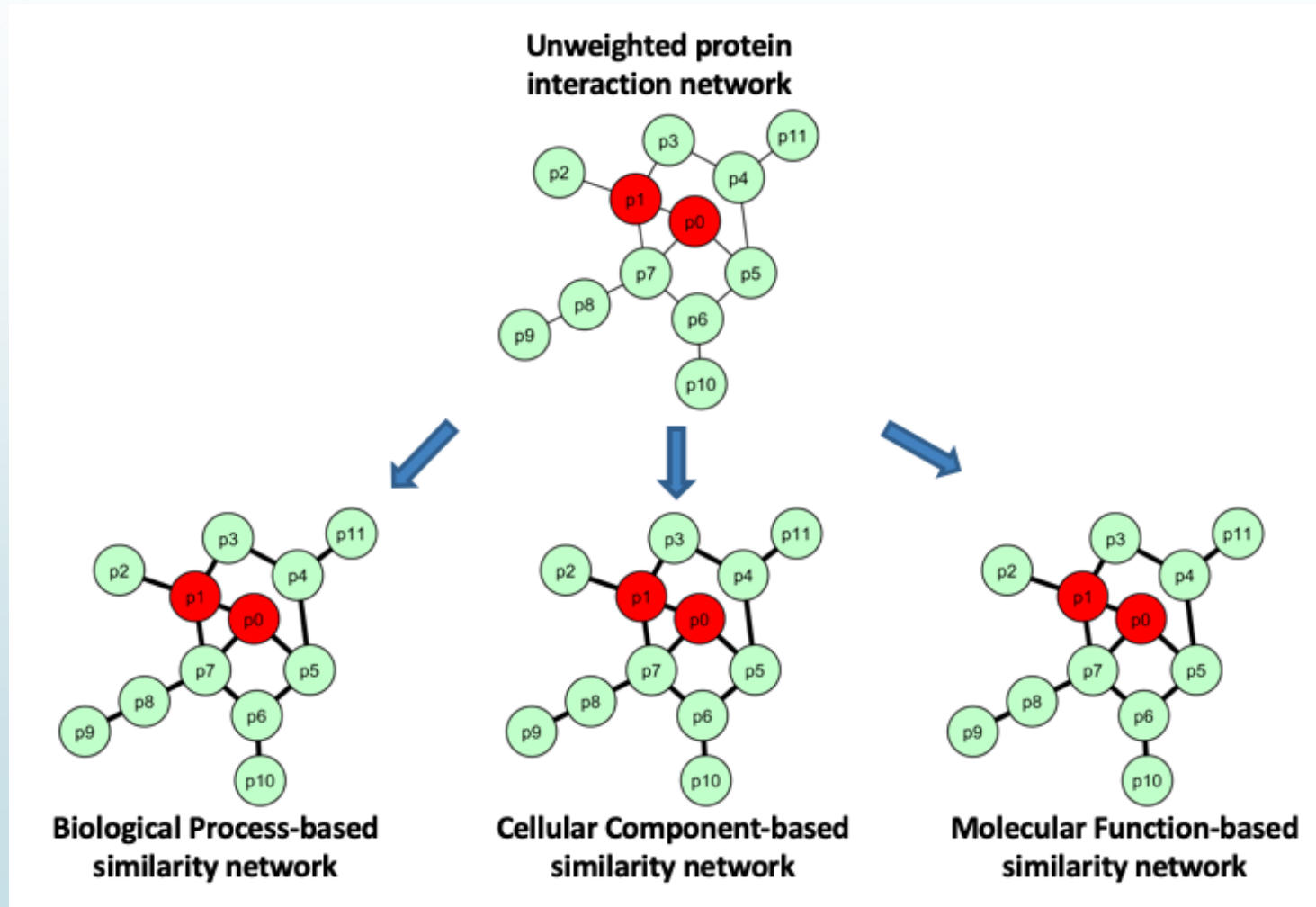


Détection communautaire

- Le domaine de la détection communautaire vise à identifier des groupes d'individus ou d'objets hautement connectés à l'intérieur de ces réseaux, ces groupes sont appelés communautés.



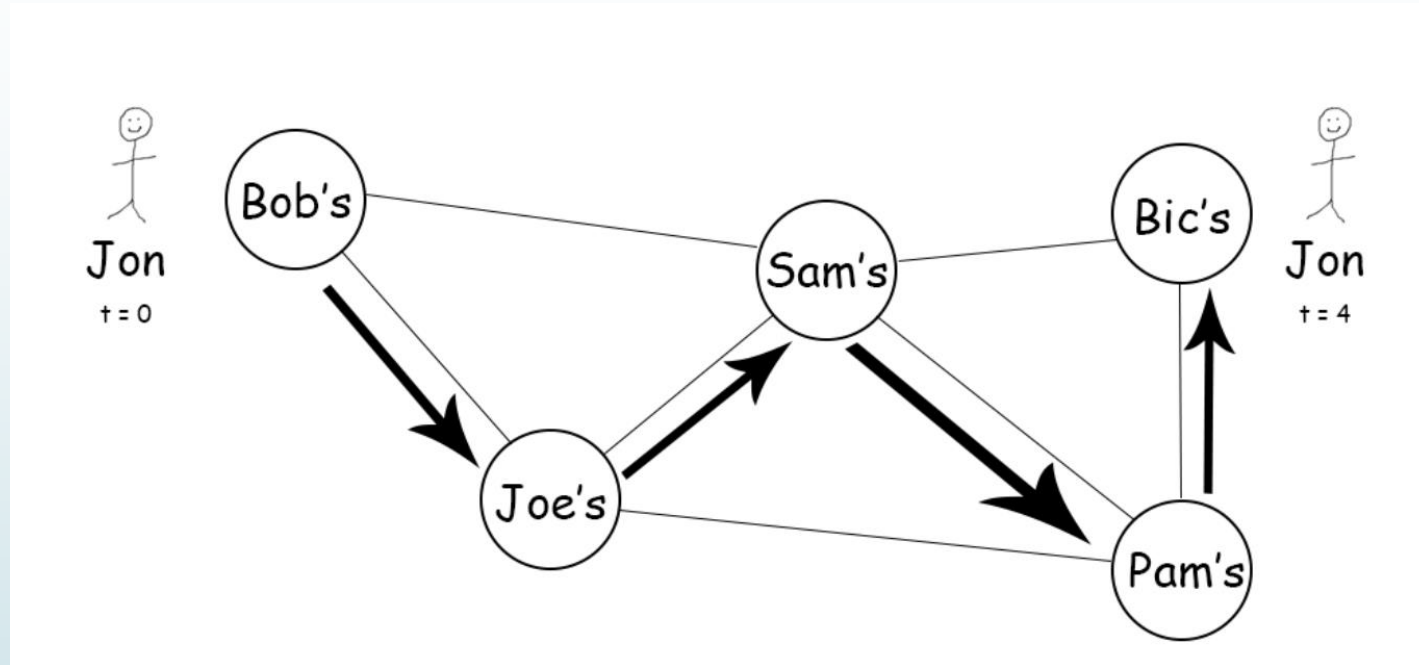
Similitude réseau





Notions de base sur les graphes

Parcourir un graphe



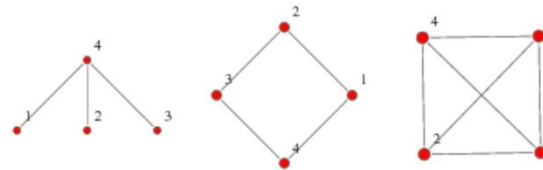
- **Une Marche**: Une suite de noeuds adjacents— une marche fermée se produit lorsque le noeud de destination est le même que le noeud source.
- **Sentier**: Une marche sans arrêt répétés — un circuit est un sentier fermé
- **Chemin**: Une marche sans noeuds répétés — un cycle est un chemin fermé

Matrice d'adjacence

- La matrice d'adjacence d'un graphe est composée de 1 et de 0, à moins qu'elle ne soit pondérée ou étiquetée autrement. A peut être construit en suivant cette règle :

$$a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

La matrice d'adjacence d'un graphe **non dirigé** est donc symétrique.



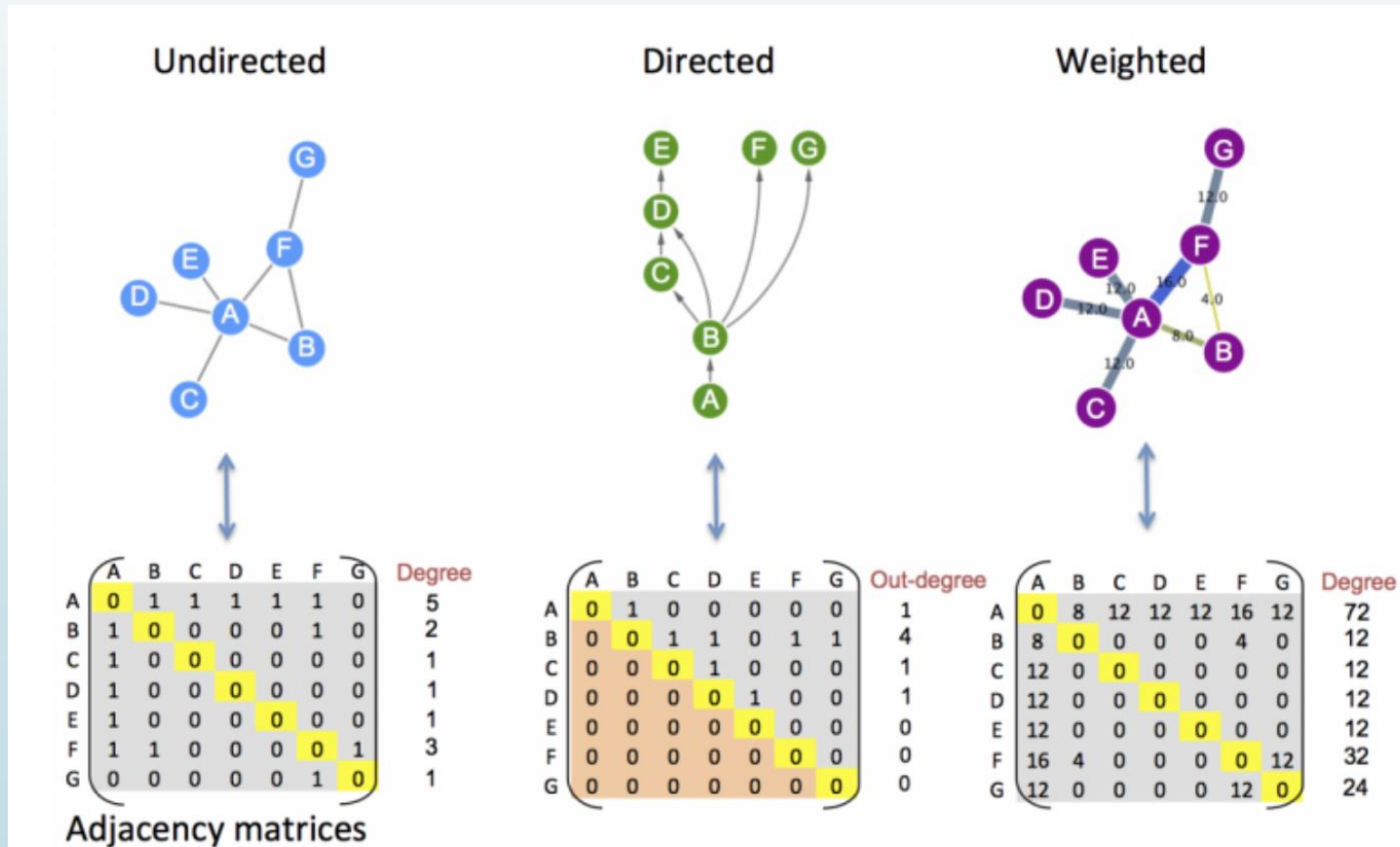
$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Matrice de degrés

- La matrice de degrés **D** d'un graphe est essentiellement une matrice diagonale, où chaque valeur de la diagonale est le degré de son nœud correspondant.

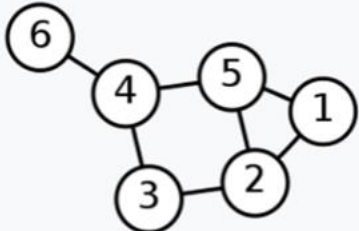


Matrice laplacienne

- La matrice laplacienne d'un graphe est le résultat de la soustraction de la matrice d'adjacence de la matrice de degrés:

$$L = D - A$$

- Chaque valeur de la matrice de degrés est soustraite par sa valeur respective dans la matrice de contiguïté en tant que telle:

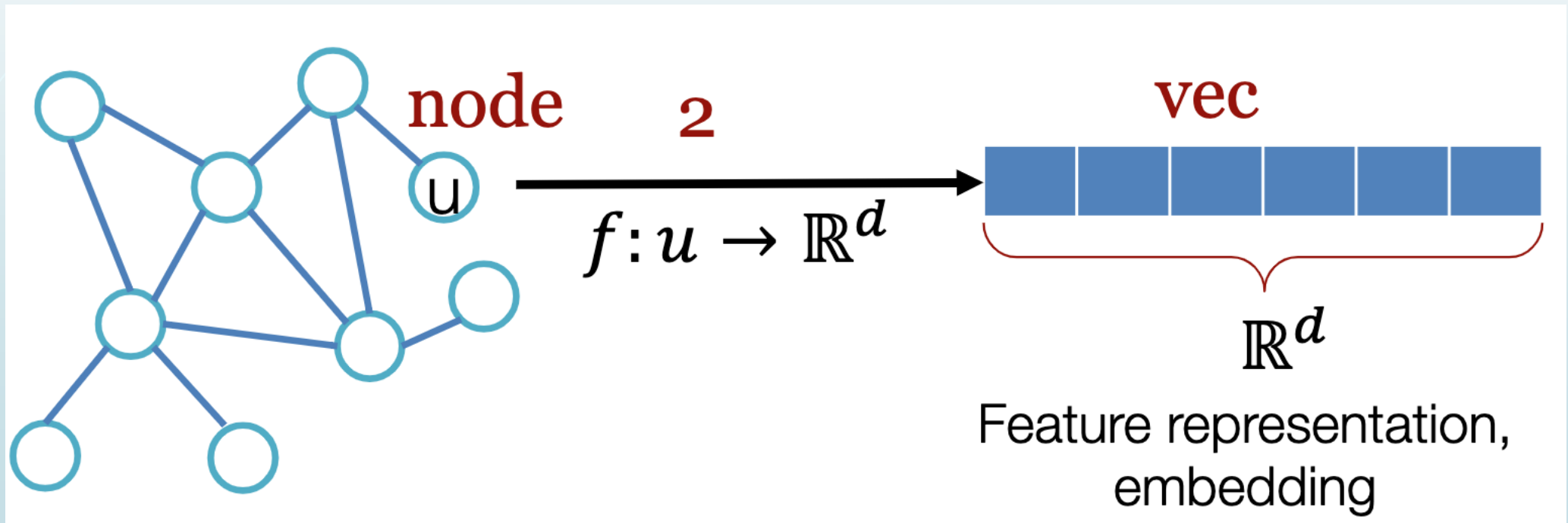
Labeled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$



Apprentissage dans les graphes

Apprentissage de caractéristiques dans les graphes

- But: apprentissage de bonne representation vectorielles pour les noeuds d'un graphe (similaire aux plongement de mots)



Representation

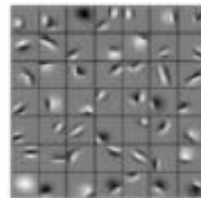
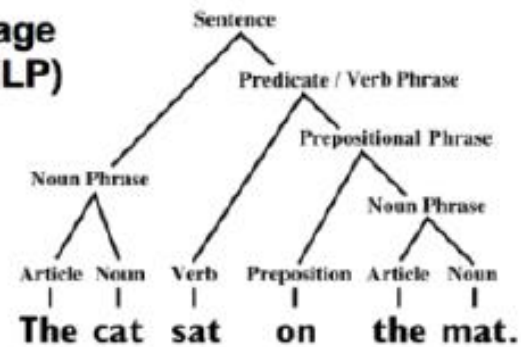
Speech data



Grid games



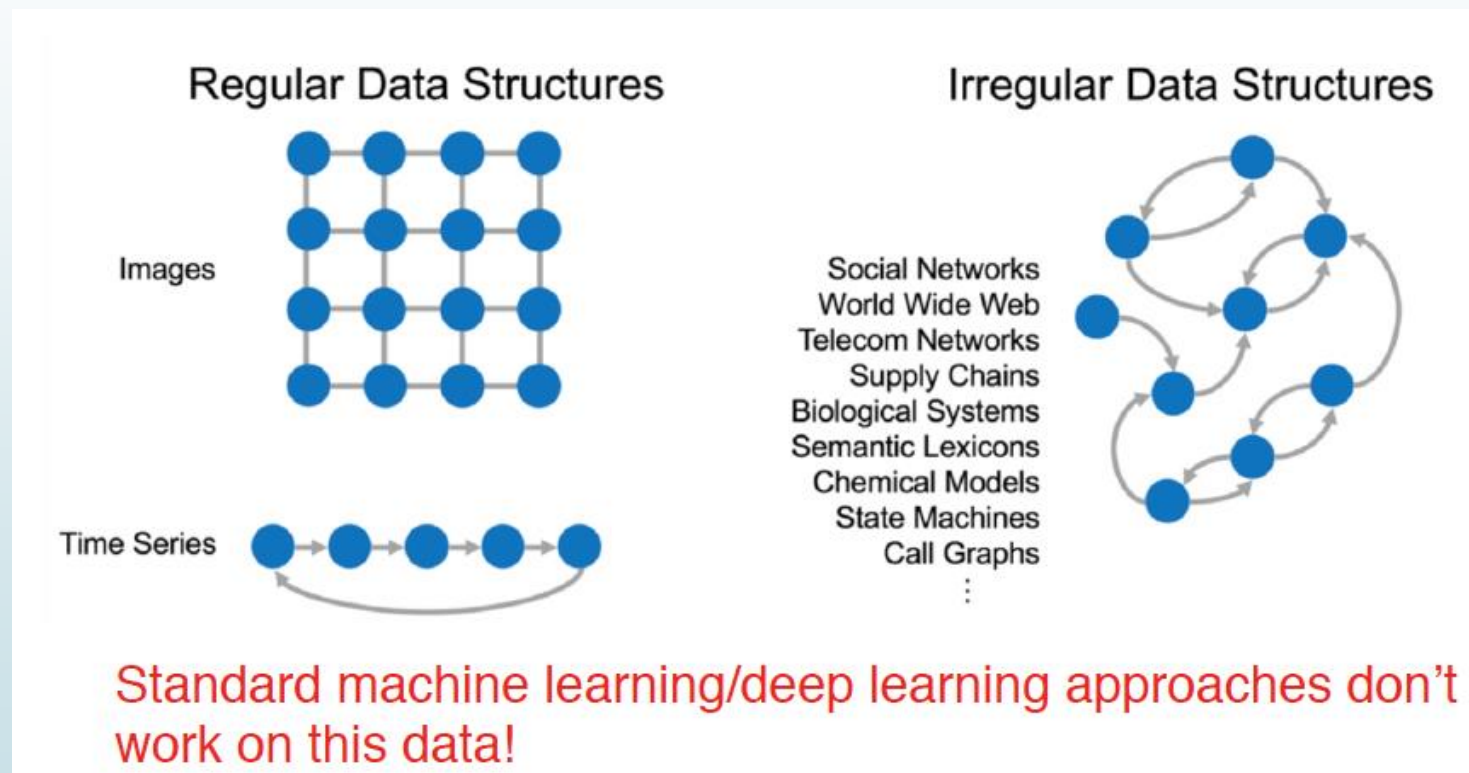
Natural language processing (NLP)



Pourquoi est-ce que l'apprentissage avec des graphes est difficile ?

Problème 1: manque de régularité dans la structure.

Difficile d'extraire systématiquement de bonnes caractéristiques

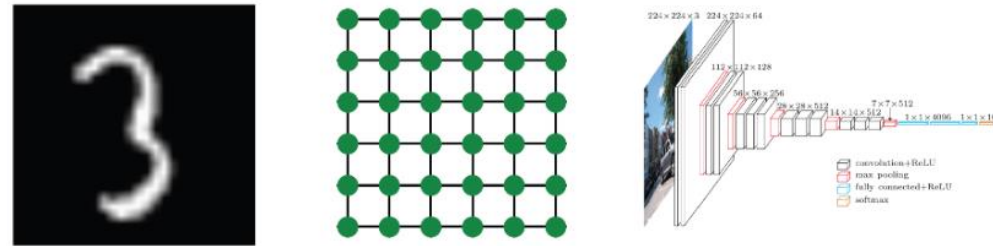


Les approches standard ne **fonctionnent pas** sur les graphes à cause de leur **manque de régularité**

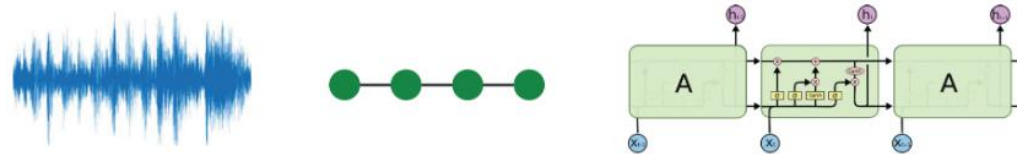
Pourquoi l'apprentissage avec des graphes est-il difficile ?

- La boîte à outils moderne d'apprentissage profond est conçue pour des séquences ou des grilles simples.

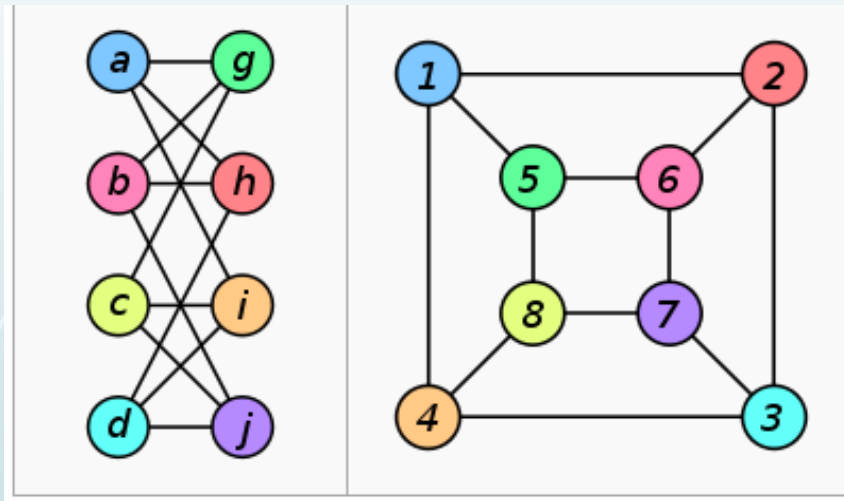
CNNs for fixed-size images/grids....



RNNs or word2vec for text/sequences...



Probleme 2



Definition:

- un graphe est dit **biparti** si son ensemble de sommets peut être divisé en deux sous-ensembles disjoints U et V tels que chaque arête ait une extrémité dans U et l'autre dans V .

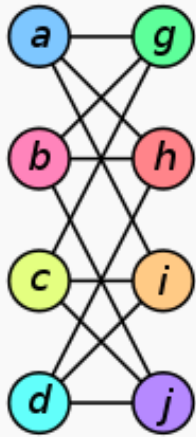
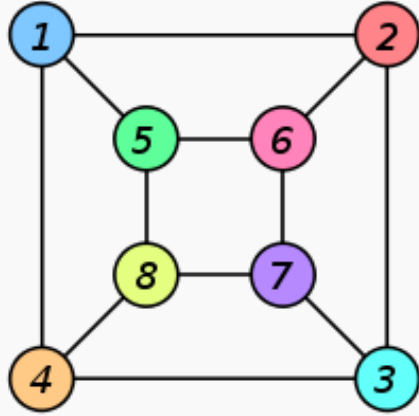
Important pour certaines application:

- Recommandation de contenu (U : contenu, V usagers) .
- Question:
Quel graphe est **biparti**?

Le problème de l'isomorphisme des graphes

- Le problème de l'isomorphisme des graphes est le problème de **décision** consistant à **déterminer si deux graphes finis sont isomorphes**.

c'est-à-dire s'ils sont les mêmes, quitte à renommer les sommets.

Graph G	Graph H	An isomorphism between G and H
		$\begin{aligned} f(a) &= 1 \\ f(b) &= 6 \\ f(c) &= 8 \\ f(d) &= 3 \\ f(g) &= 5 \\ f(h) &= 2 \\ f(i) &= 4 \\ f(j) &= 7 \end{aligned}$

- Le problème de l'isomorphisme des graphes n'est ni connu pour être ni complétement connu pour être traitable en temps polynomial. (c'est-à-dire que nous ne savons pas s'il est "dur" ou non en général).
- Néanmoins pour les curieux, il existe un algorithme en temps quasi polynomial:

$$2^{O(\log(n)^3)} \text{ (voir } \text{https://arxiv.org/pdf/1710.04574})$$



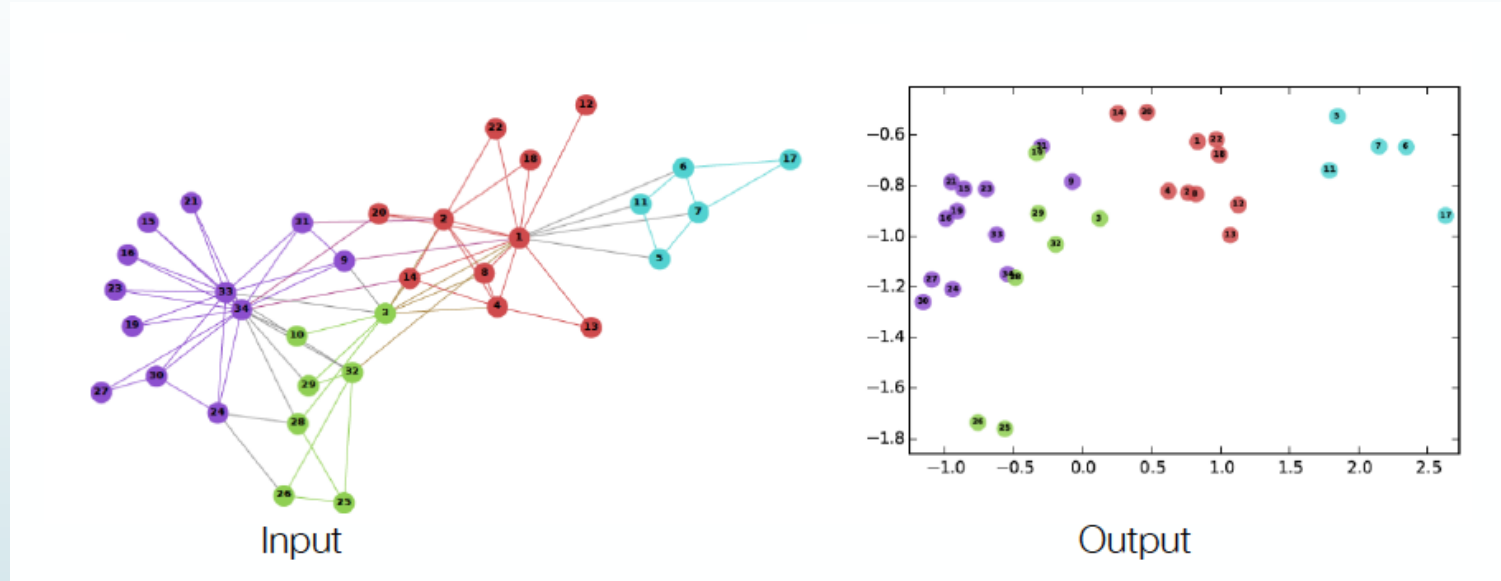
Pourquoi l'apprentissage dans les graphes est difficile ?

- Les graphes sont beaucoup plus complexes que le texte ou les données visuelles!
- Structure topographique complexe (c.-à-d. pas de localité spatiale comme les grilles)
- Pas d'ordre de nœud fixe ou de point de référence (c'est-à-dire le problème d'isomorphisme)
- Souvent dynamique et doté de fonctionnalités multimodales.
- Ils ont des invariants complexes (isomorphisme)



Plongement de noeuds

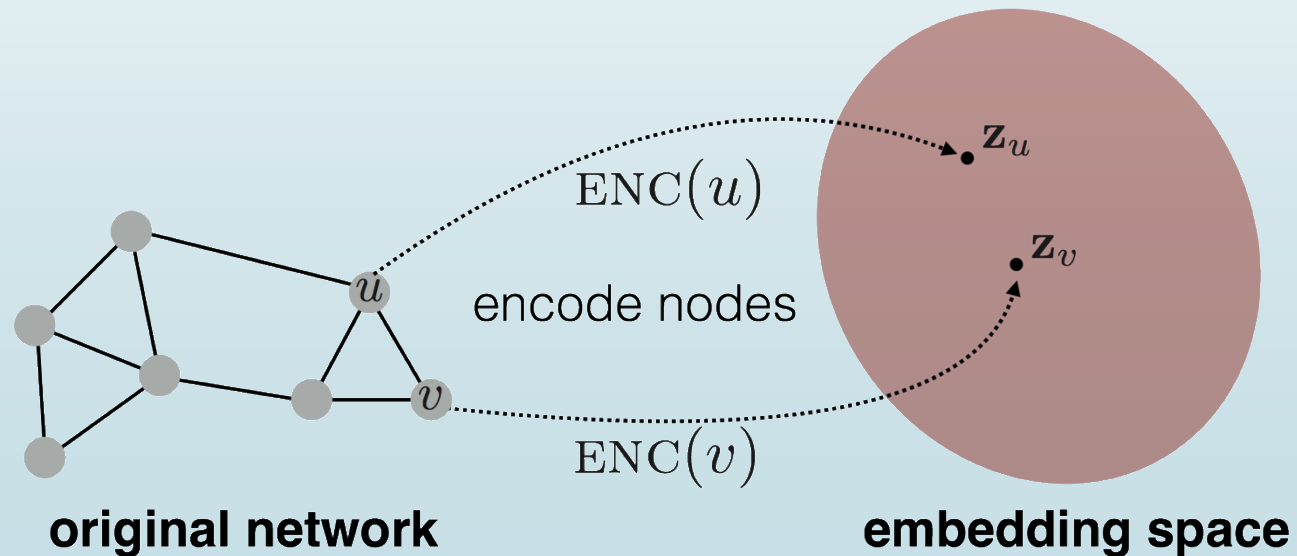
Plongement de noeuds



- **Intuition:** Recherchez le *plongement de noeuds* en dimension d afin que les noeuds « similaires » du graphique aient des vecteurs proches les uns des autres.

Plongement de noeuds

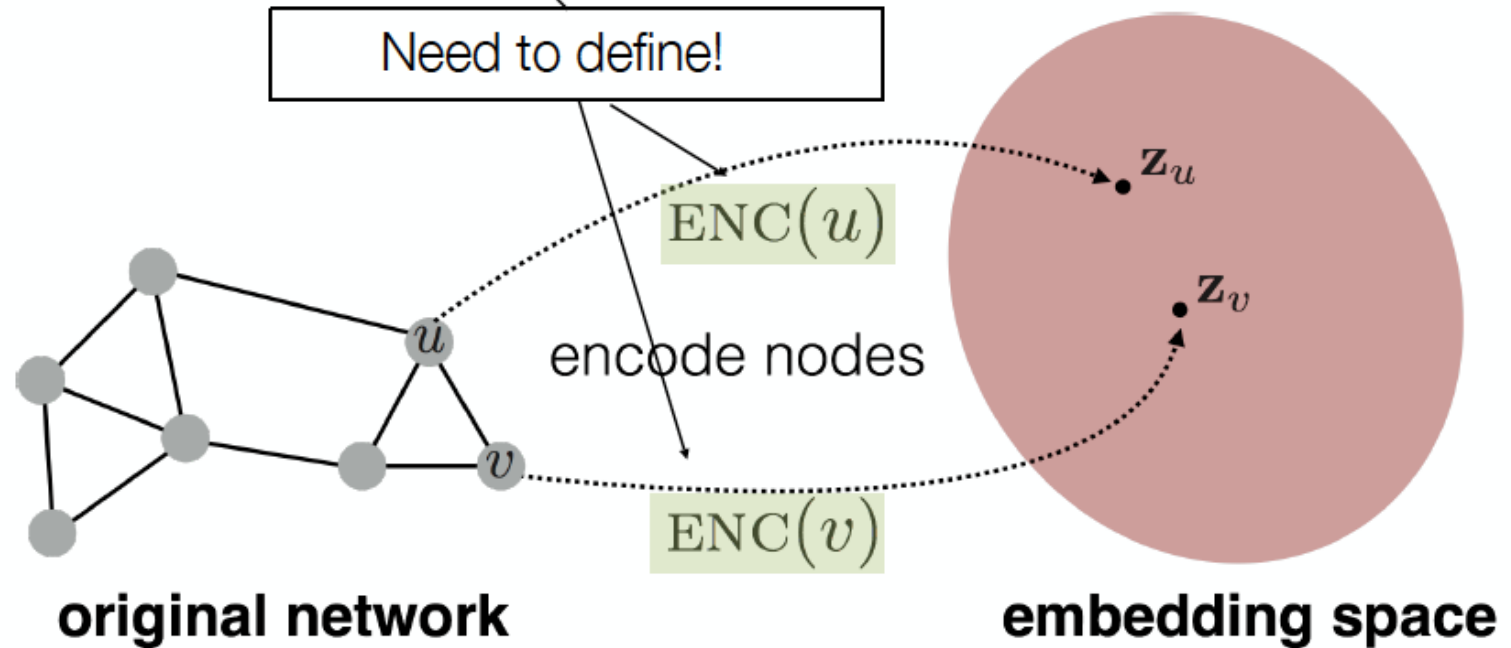
- Supposons que nous ayons un graphe G :
 - V est l'ensemble des sommets.
 - A est la matrice d'adjacence (supposons binaire).
 - Aucune caractéristique de nœud ou information supplémentaire n'est utilisée.**
- L'objectif est d'encoder les nœuds de sorte que la similitude dans l'espace de plongement (par exemple, le produit scalaire) se rapproche de la "similitude" dans le réseau d'origine.



Plongement de noeuds

Goal: $\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$

Need to define!



Plongement de noeuds - apprentissage

- Définir un encodeur (c'est-à-dire une fonction des nœuds aux plongements)
- Définir une fonction de similitude de nœud (c.-à-d. une mesure de similitude dans le réseau d'origine).
- Apprendre les paramètres de l'encodeur de sorte que :

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Composants clés

- L'encodeur associe à chaque nœud un vecteur de “faible” dimension.

$$\text{ENC}(v) = \mathbf{z}_v$$

node in the input graph

d-dimensional embedding

- La fonction de similitude spécifie comment les relations dans l'espace vectoriel correspondent aux relations du réseau d'origine.

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Similarity of u and v in the original network

dot product between node embeddings

Encodage peu profond

- Approche d'encodage la plus simple : l'encodeur n'est qu'une matrice

$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

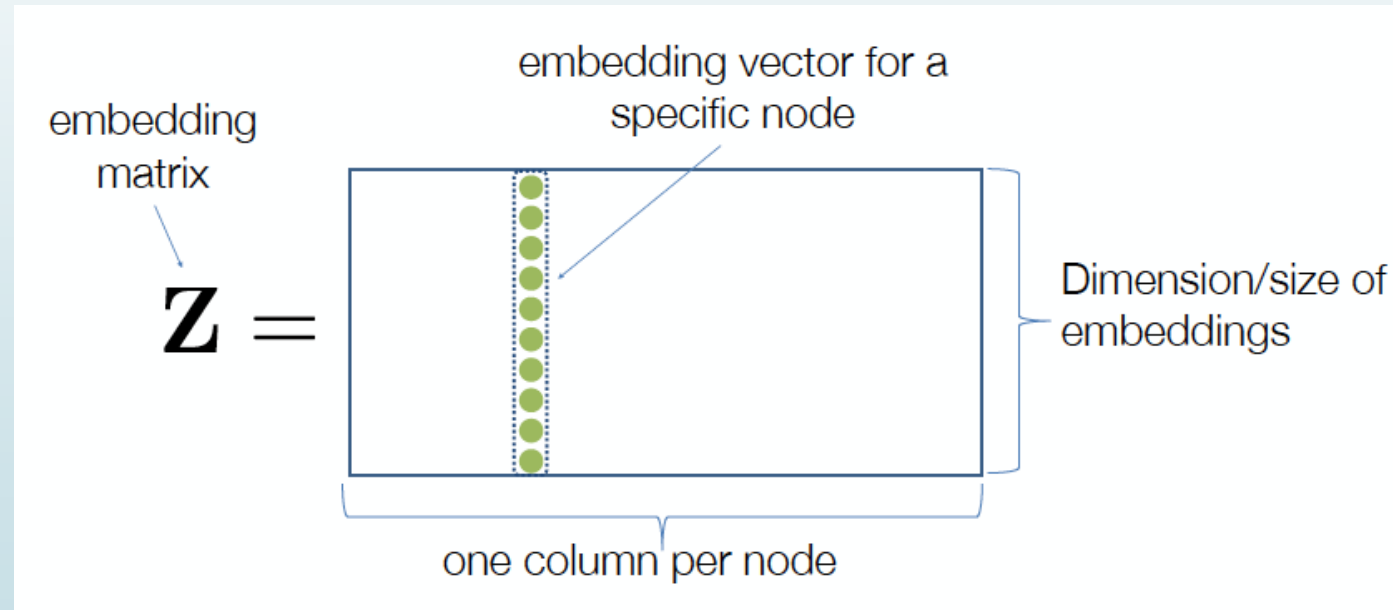
$$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|} \quad \text{matrix, each column is node embedding [what we learn]}$$

$$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|} \quad \text{indicator vector, all zeroes except a one in column indicating node } v$$

Comme le plongement lexical!

Encodage peu profond

- Approche d'encodage la plus simple: l'encodeur n'est qu'une liste des plongement de chaque noeuds.





From Shallow to Deep

Limitations de l'encodage peu profond :

- $O(|V| * d)$ paramètres sont nécessaires : il n'y a pas de partage de paramètres et chaque nœud a son propre vecteur.
- Intrinsèquement « transductif » : **il est impossible de générer des représentations pour des nœuds qui n'ont pas été vus pendant l'entraînement.**
- **N'incorpore pas de caractéristiques des nœuds** : De nombreux graphes ont des nœuds avec des caractéristiques que nous pouvons et devons exploiter.

Du peu profond à plus profond

- Nous discuterons le cours prochain de méthodes « plus profondes » basées sur des réseaux de neurones pour les graphes.

$$\text{ENC}(v) = \text{complex function that depends on graph structure.}$$

En général, tous les encodeurs peuvent être combinés avec les fonctions de similitudes qui dépendent de la structure du graphe.



Comment définir la similitude de nœud?

- La distinction clé entre les différentes méthodes « superficielles » est la façon dont elles définissent la similitude des nœuds.
- Par exemple, deux nœuds devraient-ils avoir des plongements similaires s'ils...
- sont connectés? (Similitude basée sur la contiguïté)
- partagent des voisins? (Similitude multi-sauts)
- ont des « rôles structurels » similaires ? (marches aléatoires)
- ...?



Comment définir la similitude de nœud?

- Similitude basée sur la contiguïté
- Similitude multi-sauts
- Similitudes via des marches aléatoires



Similitude basée sur la contiguïté

Similitude basée sur la contiguïté

- La fonction de similarité est juste le poids de l'arrête entre u et v dans le réseau d'origine.
- Intuition : les produits scalaires entre les plongements de nœuds approximent leur connection (i.e. le poids des arrêtes.)

The diagram illustrates the loss function \mathcal{L} for node proximity similarity. It is defined as the sum over all node pairs $(u, v) \in V \times V$ of the squared difference between the embedding similarity $\mathbf{z}_u^\top \mathbf{z}_v$ and the (weighted) adjacency matrix entry $\mathbf{A}_{u,v}$.

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \left\| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v} \right\|^2$$

Annotations:

- \mathcal{L} : loss (what we want to minimize)
- $\sum_{(u,v) \in V \times V}$: sum over all node pairs
- $\mathbf{z}_u^\top \mathbf{z}_v$: embedding similarity
- $\mathbf{A}_{u,v}$: (weighted) adjacency matrix for the graph

Similitude basée sur la contiguïté

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}\|^2$$

Trouvez une matrice de plongement qui minimise la perte

Option 1 : Utiliser la descente de gradient stochastique (SGD) comme méthode d'optimisation générale.

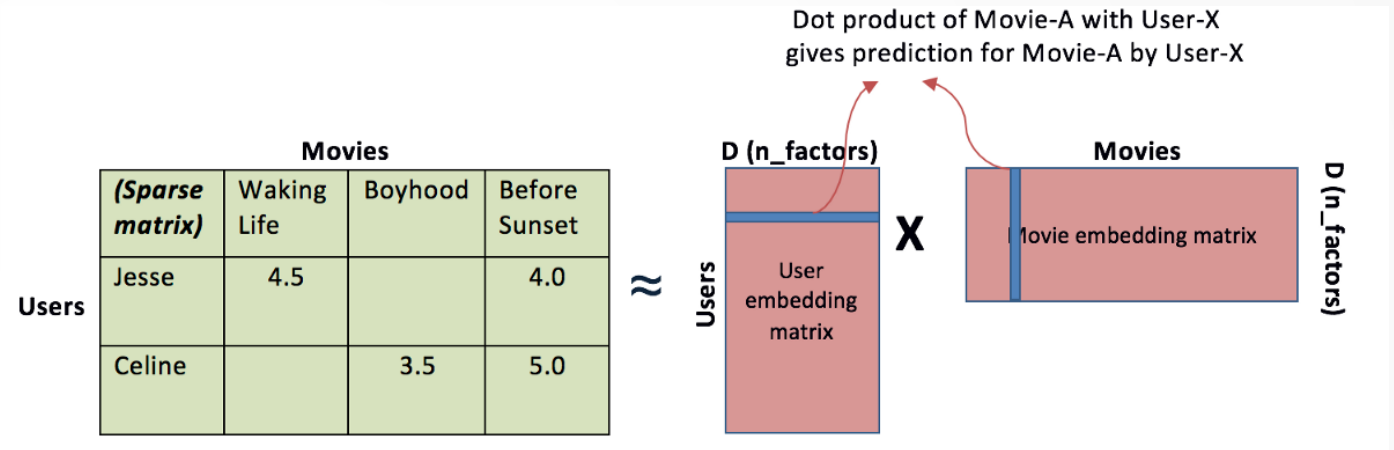
- Approche générale hautement flexible

Option 2 : Résoudre une décomposition matricielle (p. ex., SVD).

- Ne fonctionne que dans des cas limités (lorsqu'on a toutes les valeurs pour A).
- Coûteux: $O(|V|^3)$

Factorisation matricielle

- Les algorithmes de factorisation matricielle fonctionnent en décomposant la matrice d'interaction de l'élément utilisateur en un produit de deux matrices rectangulaires de dimensionnalité inférieure.
- Cette famille de méthodes est devenue largement connue lors du défi du prix Netflix 2006 en raison de son efficacité dans les systèmes de recommandation.
- Graphe Bipartite (U: users, V: Films)
- Un plongement pour chaque (U et V)
- Appris en minimzant les moindres carrés.



$$\mathcal{L} = \sum_{(u,v) \in U \times V} \| \mathbf{z}_u^T \mathbf{\bar{z}}_v - \mathbf{A}_{u,v} \|^2$$

loss (what we want to minimize)

sum over all node pairs

embedding similarity

(weighted) adjacency matrix for the graph

Similitude basée sur la contiguïté

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}\|^2$$

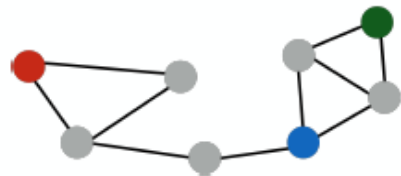
Inconvénients:

Exécution: $O(|V|^2)$. Doit prendre en compte toutes les paires de nœuds.

Peut faire $O(|E|)$ en additionnant uniquement sur des arêtes non nulles et en utilisant une régularisation

Paramètres: $O(|V| * d)$! Un vecteur appris par nœud.

Ne prend en compte que les connexions directes et locales.



e.g., the blue node is obviously more similar to green compared to red node, despite none having direct connections.



Similarité multi-sauts

Matériel basé sur:

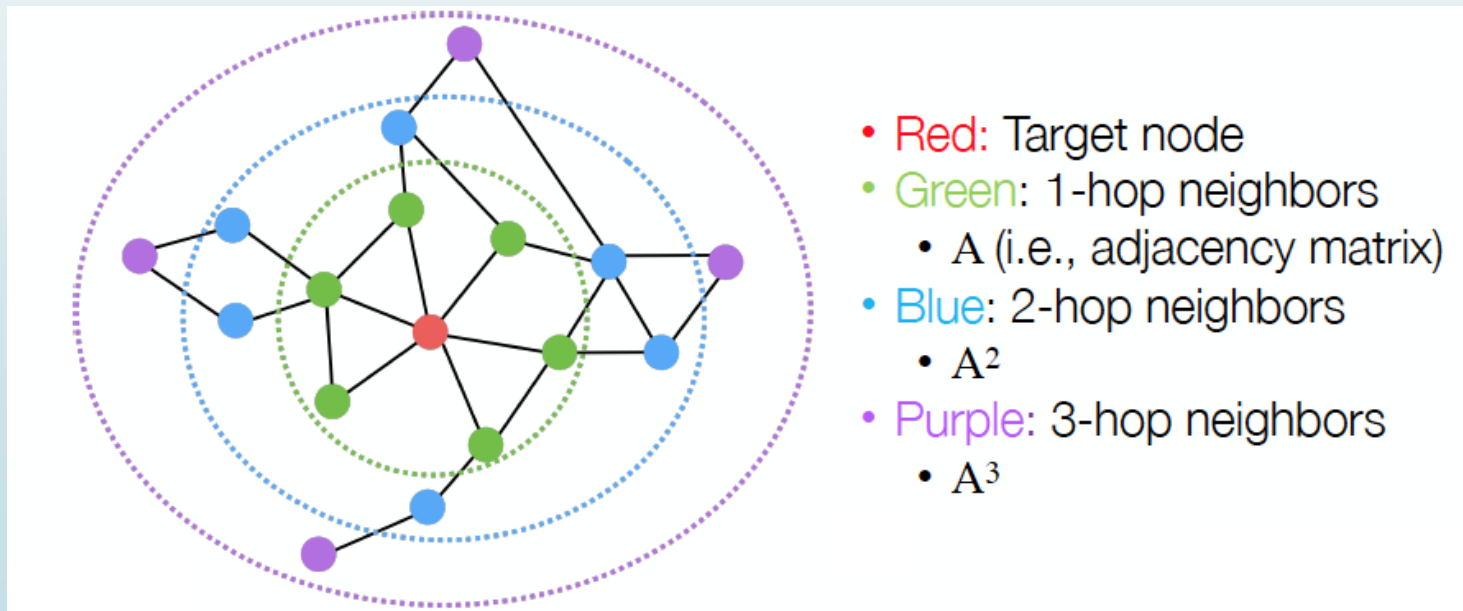
Cao et al. 2015. [GraRep: Learning Graph Representations with Global Structural Information](#)

► Ou et al. 2016. [Asymmetric Transitivity Preserving Graph Embedding](#)

► Jian Tang et al. 2015. [LINE: Large-scale Information Network Embedding](#)

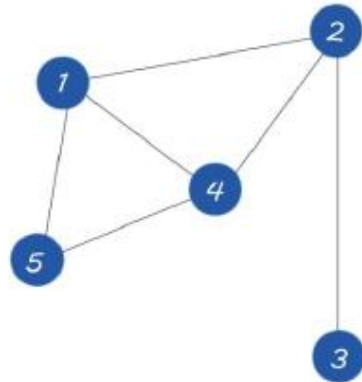
Similarité multi-sauts

- Idée: Considérez les voisinages k-sauts du nœud.
- Par exemple:



Puissances de la matrice d'adjacence

- ➡ A^k nous donne le nombre de marches du nœud i au nœud j après k étapes.

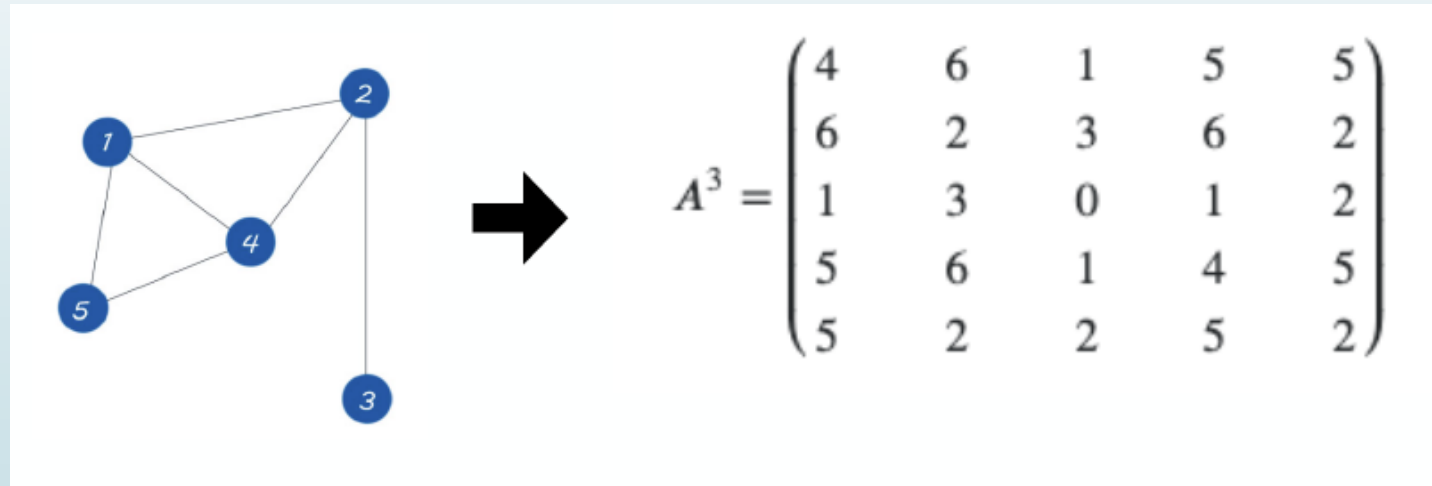


$A =$

	1	2	3	4	5
1	0	1	0	1	1
2	1	0	1	1	0
3	0	1	0	0	0
4	1	1	0	0	1
5	1	0	0	1	0

Puissances de la matrice d'adjacence

- ➡ A^k nous donne le nombre de marches du nœud i au nœud j après k étapes.



Similarité multi-sauts

- Idée de base:

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}^k\|^2$$

- Entraîner les representations à Prédire les voisins K-hop.

- En pratique (from [GraRep](#))

- Considerer les probabilités d'aller de i à j en k étapes: $P(i \rightarrow j \text{ en } k \text{ étapes}) = (A_{i,j}/d_j)^k$

- Calculer la log-proba

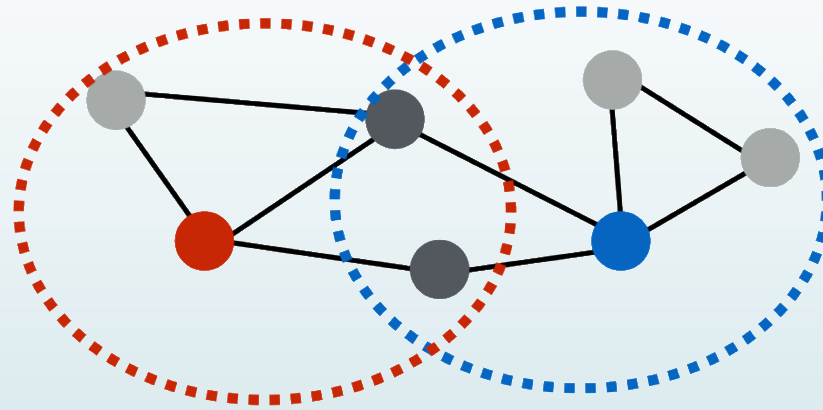
$$\tilde{\mathbf{A}}_{i,j}^k = \max \left(\log \left(\frac{(\mathbf{A}_{i,j}/d_i)^k}{\sum_{l \in V} (\mathbf{A}_{l,j}/d_l)^k} \right) - \alpha, 0 \right)$$

node degree constant shift

- Entraînez pour plusieurs longueurs de saut différentes et concaténez les plongements.

Similarité multi-sauts

- Une autre option : Mesurez le chevauchement entre les voisinages de nœuds.



Exemple de fonctions de chevauchement :

Similitude de Jaccard

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Similarité multi-sauts

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \left\| \boxed{\mathbf{z}_u^\top \mathbf{z}_v} - \boxed{\mathbf{S}_{u,v}} \right\|^2$$

embedding similarity multi-hop network similarity (i.e., any neighborhood overlap measure)

- ▶ $S_{u,v}$ est le chevauchement de voisinage entre u et v (e.g., Jaccard overlap).
- ▶ Cette technique est connue sous le nom de [HOPE](#)



Résumé

- Idée de base jusqu'à présent:
 - Définir les similitudes de nœuds par paires.
 - Optimiser les représentations de faible dimension pour vous rapprocher de ces similitudes par paires.
- problèmes:
 - Cher: En général, $O(|V|^2)$ puisque nous devons itérer sur toutes les paires de nœuds.
 - Fragile: on doit concevoir manuellement des mesures de similarité de nœuds.
 - Grand espace des paramètres (si V est grand): Paramètres: $O(|V| * d)$



Approches avec des marches aléatoires

Matériel basé sur

Perozzi et al. 2014. [DeepWalk: Online Learning of Social Representations](#)

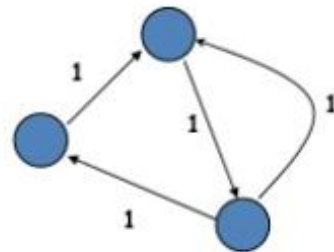
► Grover et al. 2016. [node2vec: Scalable Feature Learning for Networks](#)

Matrice de transition

Adjacency and Transition Matrix

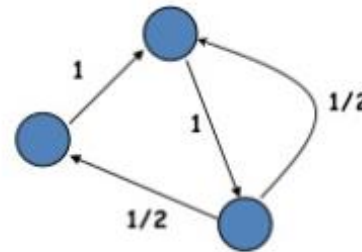
0	1	0
0	0	1
1	1	0

Adjacency matrix A

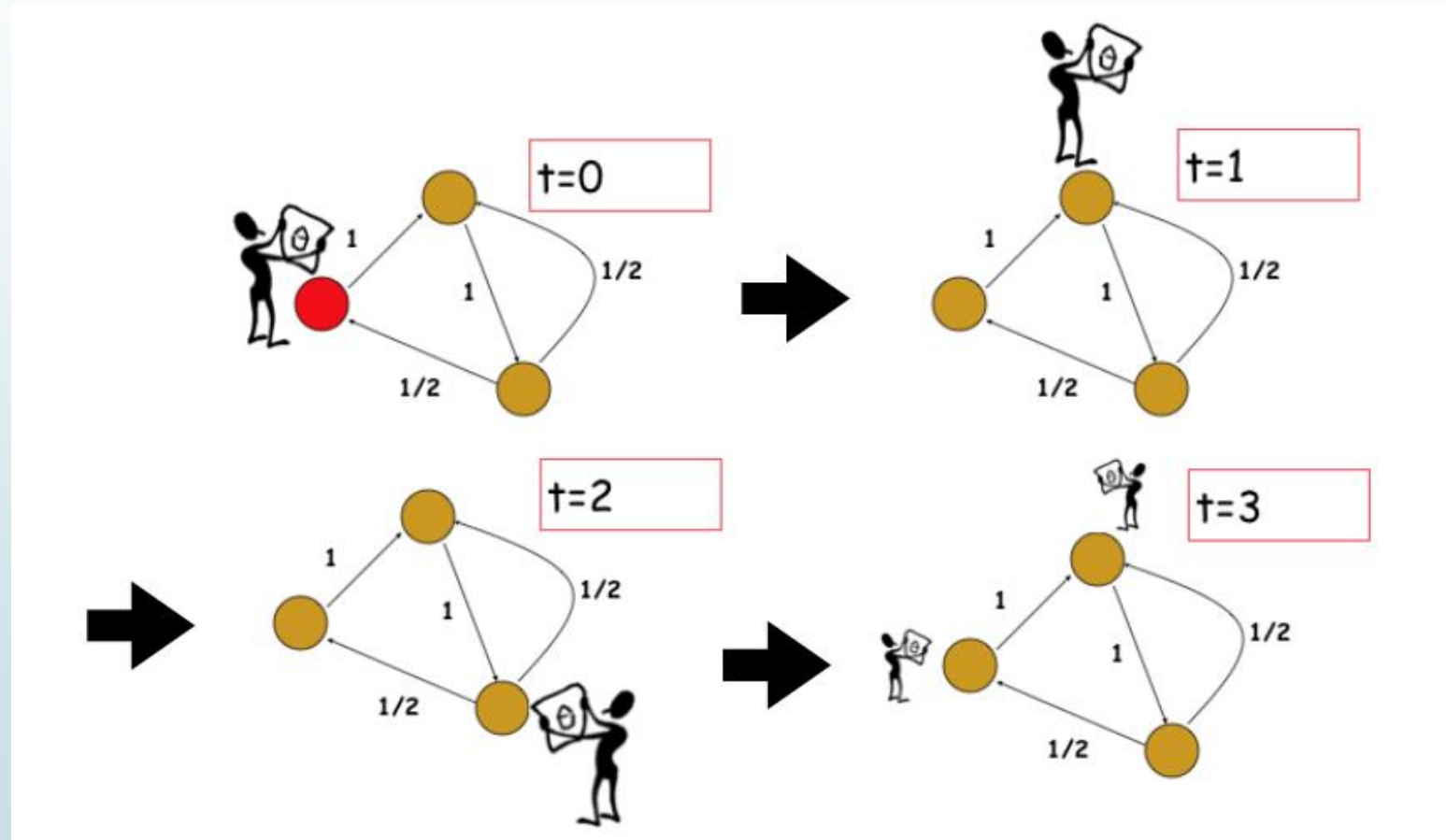


0	1	0
0	0	1
1/2	1/2	0

Transition matrix P



Marche aléatoire

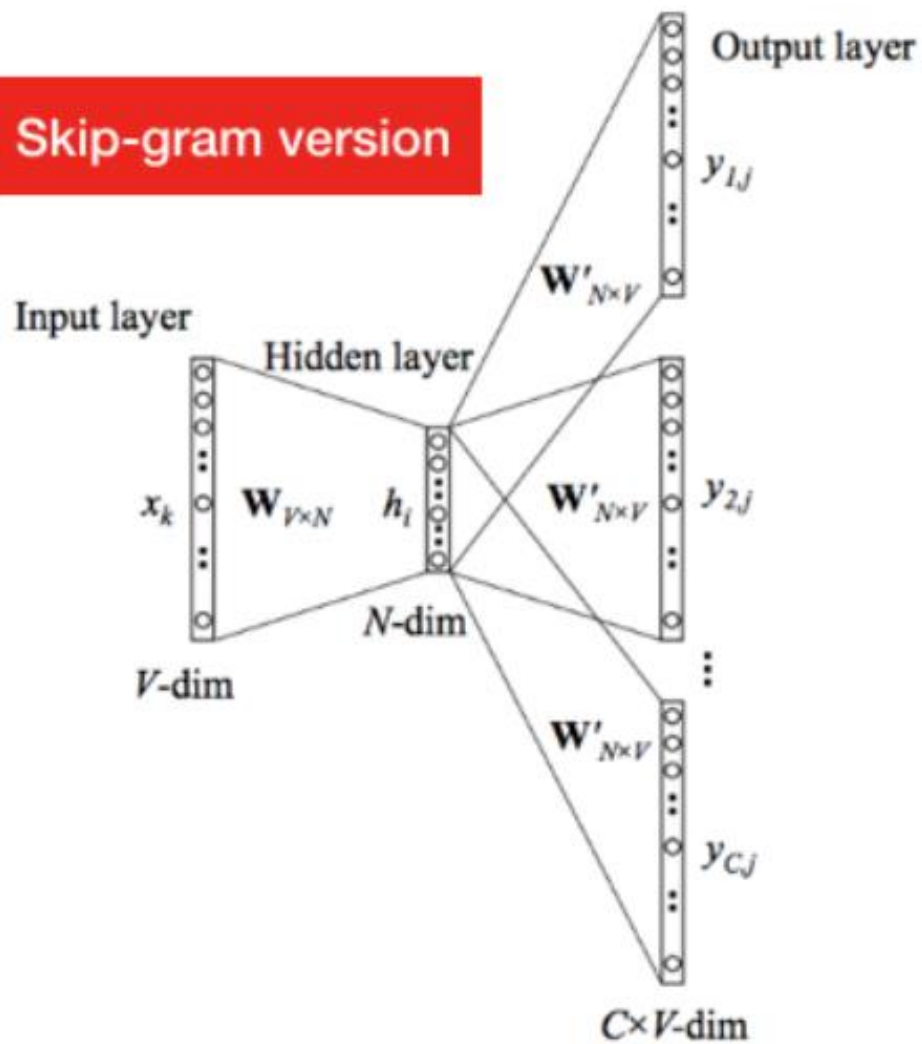


Plongement à l'aide de marche aléatoires

$$\mathbf{z}_u^\top \mathbf{z}_v \approx \text{probability that } u \text{ and } v \text{ co-occur on a random walk over the network}$$

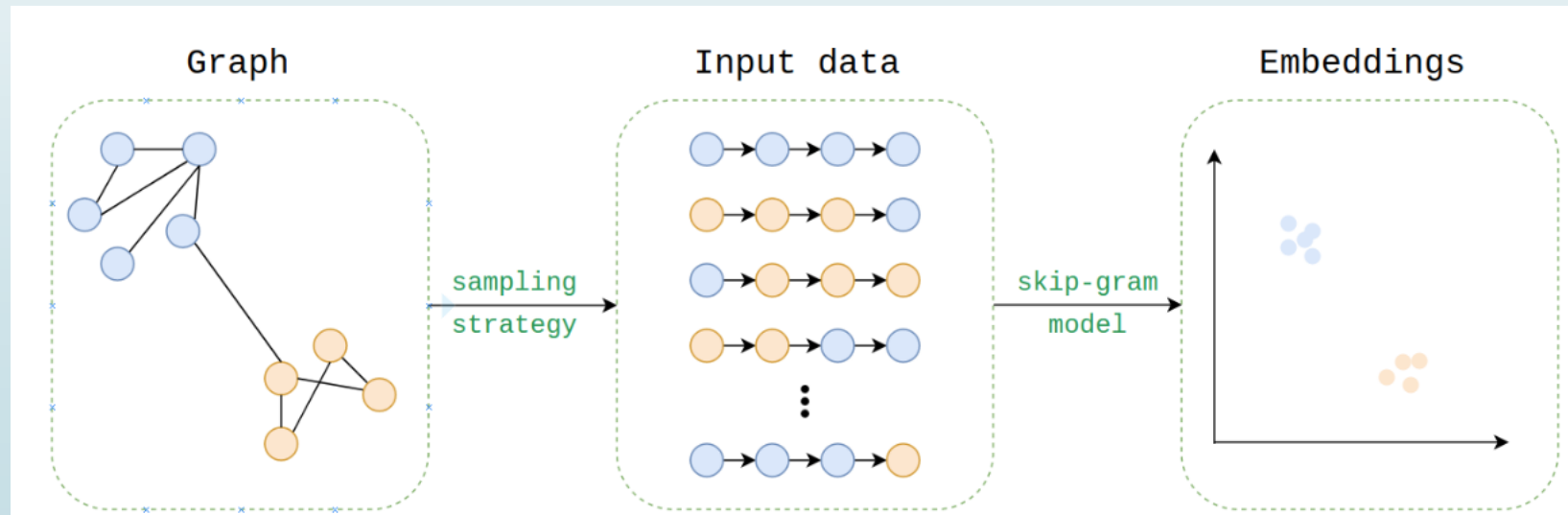
Skip-gram model

Skip-gram version

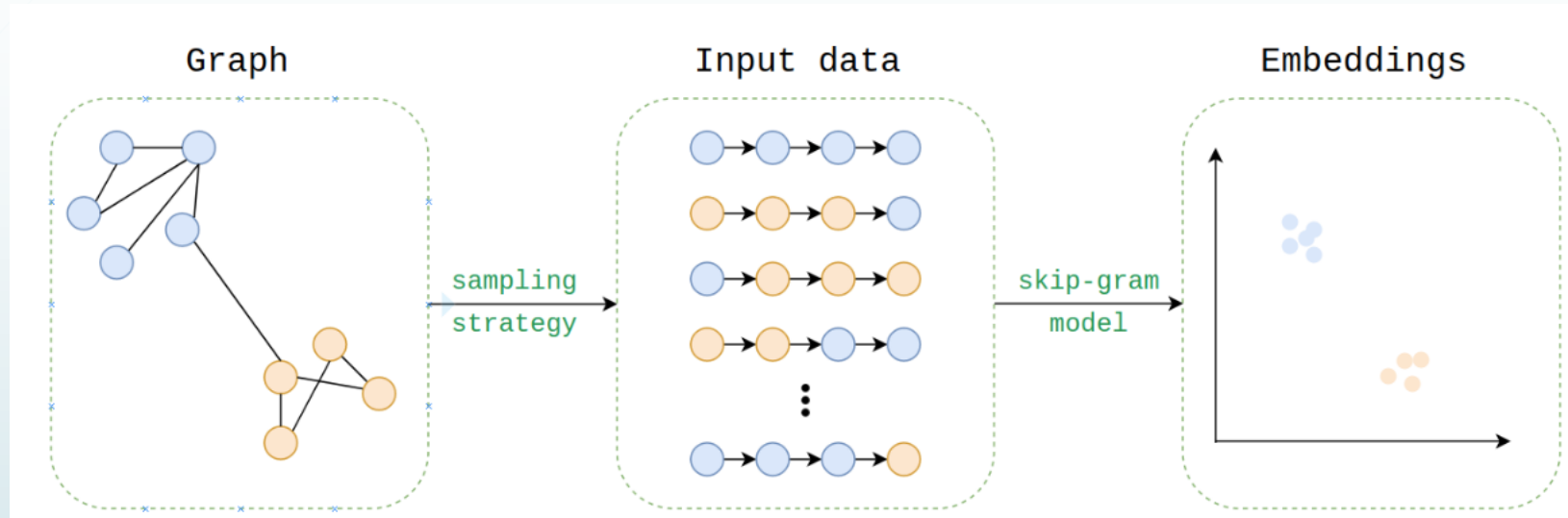


word2vec

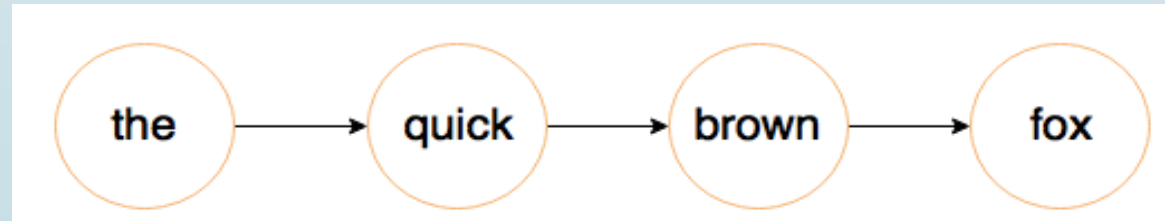
- Node2vec est similaire au modèle de skip-gram word2vec.
- De la même manière qu'un document est une séquence ordonnée de mots, on pourrait échantillonner des séquences de nœuds du réseau sous-jacent et transformer un réseau en une séquence ordonnée de nœuds.



word2vec

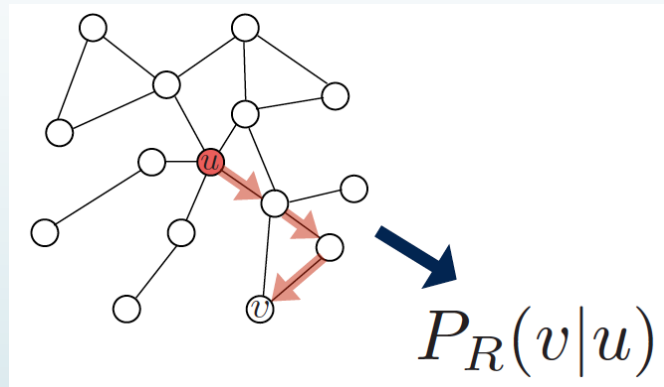


- word2vec fonctionne avec des graphes très spécifiques :

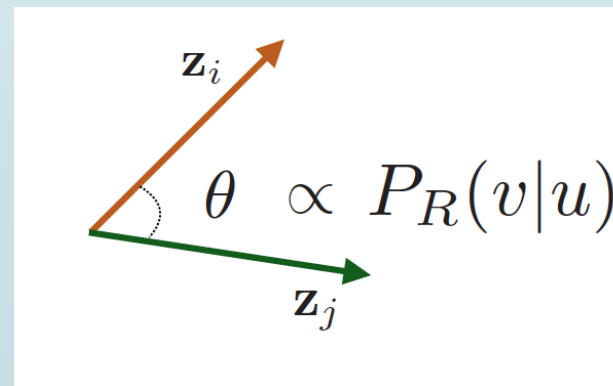


Plongement à l'aide de marche aléatoires

- Estimer la probabilité de visiter le nœud v sur une marche aléatoire à partir du nœud u en utilisant une stratégie de marche aléatoire R .



- Optimisez les intégrations pour encoder ces statistiques de marche aléatoires.





Pourquoi les marches aléatoires?

- **Expressivité**: Définition stochastique **flexible** de la similitude des nœuds qui intègre à la fois des informations de **voisinage locales** et **d'ordre supérieur**.
- **Efficacité**: Vous n'avez pas besoin de prendre en compte toutes les paires de nœuds lors de l'entraînement; Il suffit de considérer **paires qui apparaissent lors de marches aléatoires**.

Entraînement via des marches aléatoires

- ▶ Exécutez de courtes marches aléatoires à partir de chaque nœud du graphique en utilisant une stratégie R .
- ▶ Pour chaque nœud u recueillir $N_R(u)$, Le multiset* de nœuds visités lors de promenades aléatoires à partir de u .
- ▶ Entraîner les representations en fonction de :

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

* $N_R(u)$ peut avoir des éléments répétés puisque les nœuds peuvent être visités plusieurs fois lors de marches aléatoires.

Entraînement via des marches aléatoires

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

- **Intuition:** Optimisez les intégrations pour maximiser la **probabilité de cooccurrences lors d'une marche aléatoire**.
- Paramétrer $P(v | \mathbf{z}_u)$ en utilisant le softmax:

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}$$

Entraînement via des marches aléatoires

- En résumé:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} - \log \left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)} \right)$$

Diagram illustrating the components of the loss function \mathcal{L} :

- sum over all nodes u** (indicated by a red arrow pointing to $\sum_{u \in V}$)
- sum over nodes v seen on random walks starting from u** (indicated by a green arrow pointing to $\sum_{v \in N_R(u)}$)
- predicted probability of u and v co-occurring on random walk** (indicated by a blue arrow pointing to the fraction inside the log)

- Optimisation des intégrations de marche aléatoires = Recherche de représentations \mathbf{z}_u qui minimisent \mathcal{L}

Entraînement via des marches aléatoires

- Mais le faire naïvement coûte trop cher!

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log \left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)} \right)$$

Nested sum over nodes
gives $O(|V|^2)$ complexity!!

Entraînement via des marches aléatoires

- ➡ Mais le faire naïvement coûte trop cher!

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log \left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)} \right)$$

The normalization term from the softmax is the culprit... can we approximate it?

Échantillonnage négatif (voir Word2Vec!)

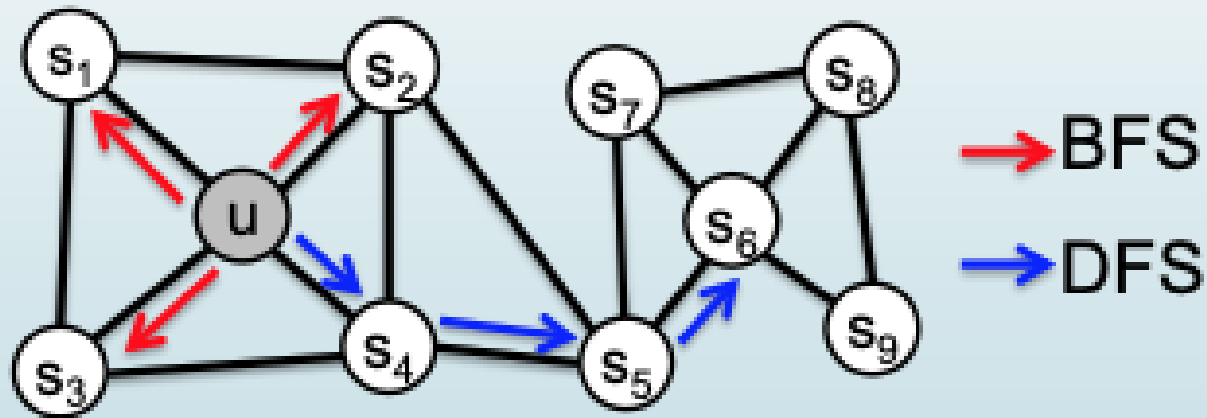


Comment devrions-nous marcher au hasard?

- Jusqu'à présent, nous avons décrit comment optimiser les intégrations en fonction des statistiques de marche aléatoires.
- Quelles stratégies devrions-nous utiliser pour exécuter ces marches aléatoires?
- L'idée la plus simple : il suffit d'exécuter des marches aléatoires de **longueur fixe sans biais** à partir de chaque nœud ([Deepwalk, 2013](#))
 - Mais pouvons-nous faire mieux?

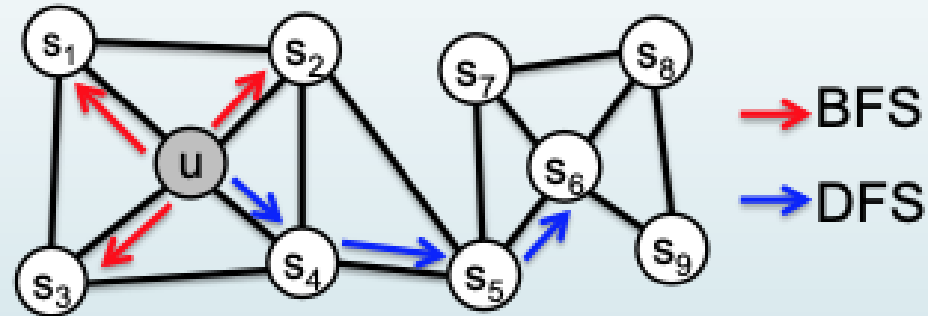
node2vec : Promenades biaisées

- Idée : utiliser des marches aléatoires flexibles et biaisées qui peuvent faire des compromis entre les vues locales et globales du réseau ([Grover and Leskovec, 2016](#))



node2vec: Biased Walks

- Deux stratégies classiques pour définir un voisinage d'un nœud donné :



$$N_{BFS}(u) = \{ s_1, s_2, s_3 \} \quad \text{Local microscopic view}$$

$$N_{DFS}(u) = \{ s_4, s_5, s_6 \} \quad \text{Global macroscopic view}$$

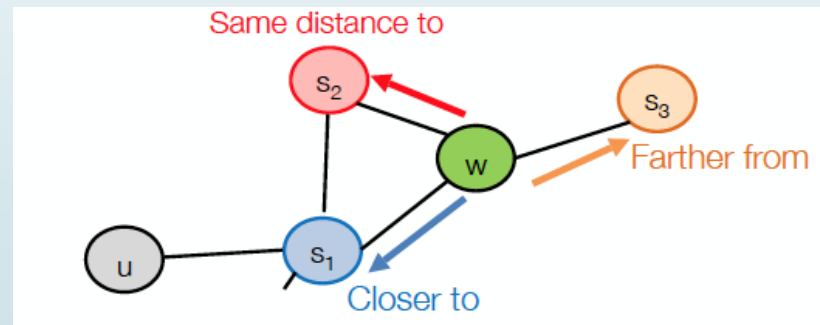
Stratégie d'échantillonnage node2vec

- ▶ La stratégie d'échantillonnage de Node2vec, accepte 4 arguments:
 - ▶ **Number of walks**: Nombre de marches aléatoires à générer à partir de chaque nœud du graphique
 - ▶ **Walk length**: Combien de nœuds y a-t-il dans chaque marche aléatoire
 - ▶ **P** : Hyperparamètre de retour
 - ▶ **q** : Hyper-paramètre d'entrée (hyper-paramètre « walk away »)

Plus les paramètres **standard skip-gram** (taille de la fenêtre contextuelle, nombre d'itérations, etc.)

Marches aléatoires biaisées

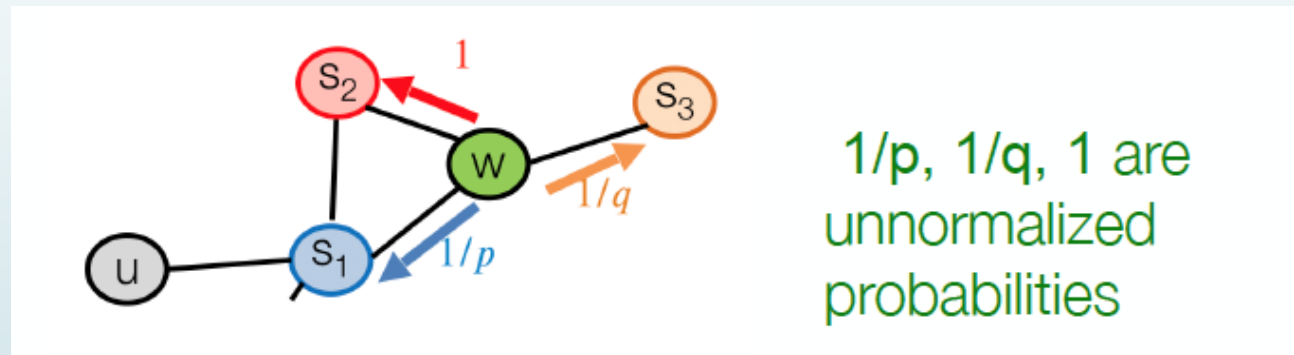
- Des promenades aléatoires biaisées de **second ordre** explorent les quartiers du graphes:
- La marche aléatoire a commencé à u et est maintenant à w
- Remarque: Les voisins de w ne peuvent être que dans 3 categories:



- Idée : se rappeler de l'origine de la marche aléatoire.

Marches aléatoires biaisées

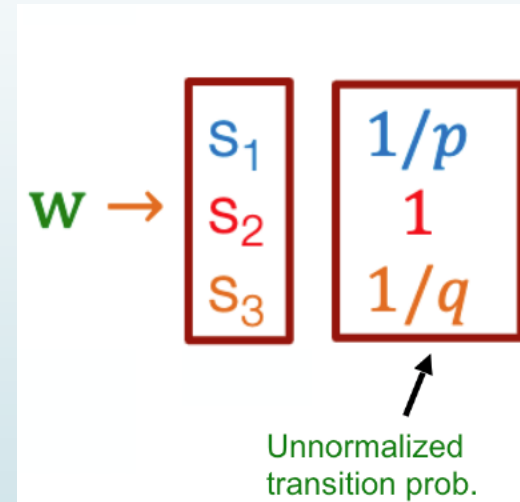
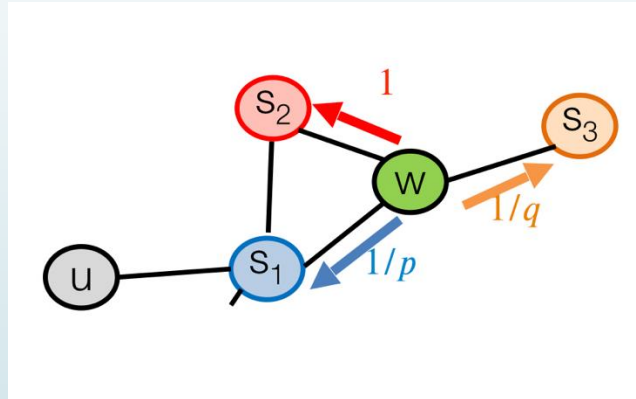
- Walker est à w . Où aller ensuite?



- p, q représentent les probabilités de transition.
 - p : paramètre de retour
 - q : Paramètre d'éloignement

Marches aléatoires biaisées

- Walker est à w . Où aller ensuite?

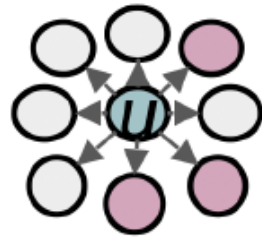


- Marche de type BFS: Faible valeur pour p
- Promenade de type DFS: Faible valeur pour q
- $Ns(u)$ sont les nœuds visités par le marcheur

BFS: Breath first search -- » parcours en largeur

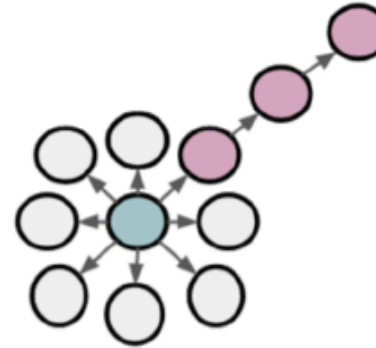
DFS: Depth first search – » parcours en profondeur

BFS vs. DFS



BFS:

Micro-view of
neighbourhood



DFS:

Macro-view of
neighbourhood



BFS: Breath first search — » parcours en largeur

DFS: Depth first search – » parcours en profondeur



Autres idées de marche aléatoire

- Différents types de marches aléatoires biaisées:
 - Basé sur les attributs de nœud ([Dong et al, 2017](#))
 - Basé sur des poids appris ([Abu-El-Haija et al., 2017](#))
- Schémas d'optimisation alternatifs:
 - Optimiser directement en fonction des probabilités de marche aléatoire de voisinage 1 saut et 2 sauts (as in [LINE from Tang et al. 2015](#))
- Techniques de prétraitement réseau :
 - Exécuter des marches aléatoires sur des versions modifiées du réseau d'origine (e.g. [Ribeiro et al. 2017's struct2vec](#), [Chen et al. 2016's HARP](#))



Résumé

- **Idée de base**: représenter les nœuds avec des vecteurs afin que les distances dans l'espace vectoriel reflètent les similitudes des nœuds dans le réseau d'origine.
- Différentes notions de similitude de nœud :
 - Basé sur la contiguïté (c.-à-d. similaire s'il est connecté)
 - Similarité multi-sauts.
 - Approches de marche aléatoires.



Alors, quelle méthode dois-je utiliser?

- ▶ Aucun gagnant unique
 - ▶ Par exemple, node2vec fonctionne mieux sur la classification des nœuds tandis que les méthodes multi-sauts fonctionnent mieux sur la prédiction de liaison ([Goyal and Ferrara, 2017 survey](#))
- ▶ Les approches de marche aléatoire sont généralement plus efficaces ($O(|E|)$ vs $O(|V|^2)$)
- ▶ Règle du pouce : choisissez une similitude de nœud adaptée à l'application