**TP2 : Les plongements basés sur les marches aléatoires**

L’objectif de cet TP est de comprendre et d’appliquer la méthode de plongement de graphe Node2vec[[1]](#footnote-1) à la tâche de classification de nœuds.

**L’algorithme Node2vec :**

La méthode de plongement Node2vec a le même objectif que la méthode de plongement que vous avez implémentée lors de la séance précédente (Voir Enoncé TP1). L’objectif est de plonger les nœuds dans un espace vectoriel, transformer les nœuds du graphe en vecteurs, de telle sorte que la similarité dans l’espace de plongement (produit scalaire) soit une bonne approximation de la similarité dans le graphe originale voir la figure 2.

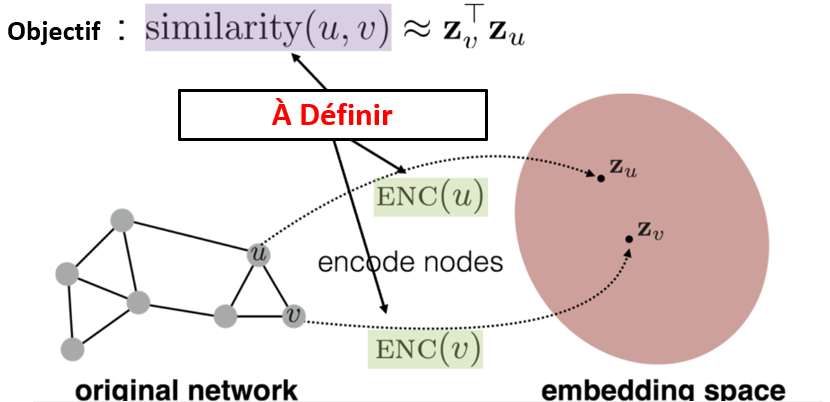
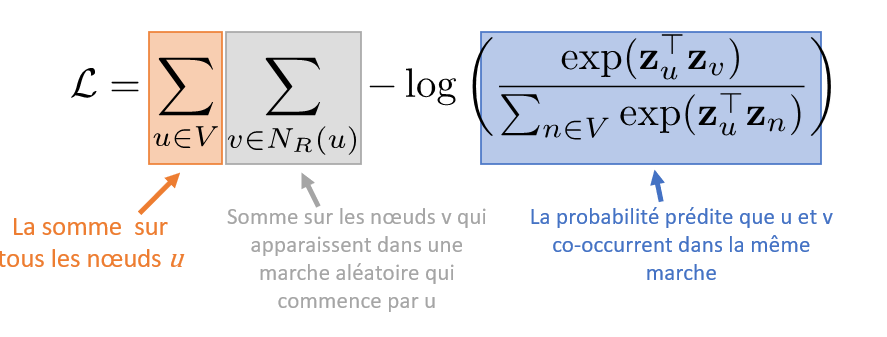


Figure 1. Rappel du principe de plongement de nœuds

La méthode Node2vec considère une similarité de nœuds basée sur les marches aléatoires. La similarité entre deux nœuds dans l’espace de graphe est la probabilité que ces derniers co-occurrents dans la même marche.

Ainsi l’objectif et de trouver des plongements qui minimisent la fonction d’erreur suivante :



La complexité de calcul de cette fonction d’erreur est d’ordre (deux sommes imbriquées et ) à cause de la normalisation (formule dans le cadre rouge) qui est coûteuse. Afin de pallier ce problème, on doit trouver une approximation précise et rapide (en termes de complexité de calcul) de la probabilité logarithmique (en bleu). La solution proposée dans Node2vec est d’utiliser un échantillonnage des marches négatives (fausses marches) (En anglais Negative Sampling). Ainsi l’approximation est donnée par la formule ci-dessous, représente le nombre de d’échantillons négatifs et est un nœud choisi aléatoirement de l’ensemble (d’une fausse marche aléatoire négative).



Quant à la stratégie des marches aléatoires, Node2vec génère des marches aléatoires flexibles et biaisées qui permet à la fois d’avoir une vue **globale** et **locale** de voisinage de chaque nœud (voir Figure 2). L’idée est de combiner les deux stratégies d’exploration de voisinage : le parcours en largeur (BFS) et le parcours en profondeur (DFS).

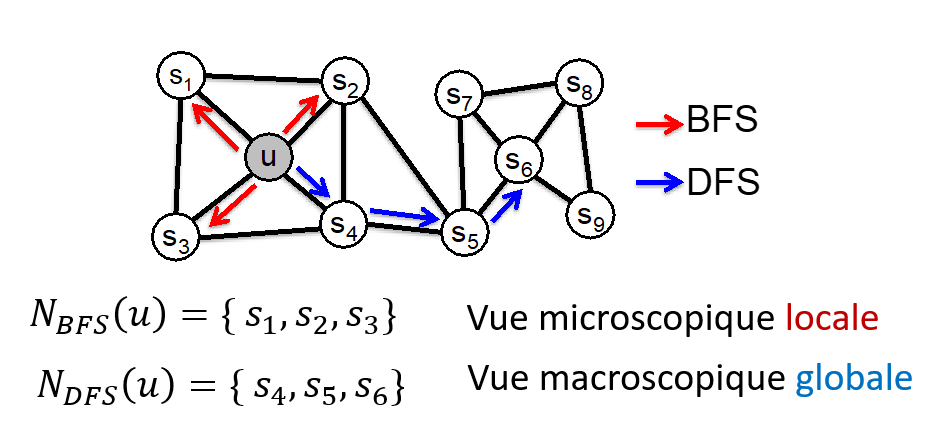


Figure 2. Stratégie des marches aléatoires de Node2vec

Pour ce faire, Node2vec utilise des marches aléatoires **biaisées de second-ordre** qui prennent compte à la fois l'état actuel et l'état précéden**t**. En d’autres termes, lorsque l'algorithme calcule les probabilités de transitions, il tient également compte de sa position précédente. Cela est assuré par deux paramètres et **.** Dans l’exemple de la figure 3, La marche vient de passer du nœud E au nœud A (la position précédente E) et évalue maintenant son prochain mouvement. La probabilité de revenir en arrière et de revisiter immédiatement un nœud de la marche est contrôlée par le paramètre de retour . Une valeur élevée pour le paramètre réduit les chances de revisiter un nœud. D'autre part, si la valeur du paramètre est petite, les chances de revenir en arrière dans la marche sont plus élevées, ce qui maintient les voisins visités plus près du nœud de départ (exploration en BFS).

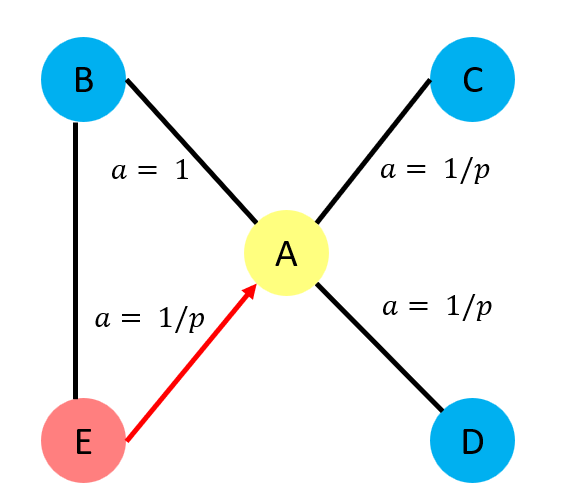


Figure 3. Les probabilités non normalisées de transition dans une marche biaisée de 2eme ordre

Le paramètre (Walk away) permet de différencier entre les nœuds de voisinage intérieur et les nœuds de voisinage extérieur. Définir une valeur élevée pour le paramètre biaise la marche aléatoire pour se déplacer vers les nœuds proches du nœud de l'étape précédente. En revenant à l'exemple de la figure précédente, si vous définissez une valeur élevée pour le paramètre , la marche aléatoire à partir du nœud A est biaisée vers les nœuds les plus proches du nœud E. De telles marches obtiennent une vue locale du graphe sous-jacent par rapport au nœud de départ (parcours en BFS). En revanche, si la valeur de est faible , la marche est plus encline à visiter des nœuds plus éloignés du nœud E. Cette stratégie encourage l'exploration vers l'extérieur et se rapproche d’une recherche DFS.

**Recommandations :**

Avant de commencer le TP, assurez-vous que vous avez installé tous les modules nécessaires (voir le fichier Requirements.txt). En cas de problème de mémoire insuffisante, vous pouvez augmenter la limite de taille mémoire utilisée par votre notebook (Jupyter) à l’aide de cette instruction :

jupyter notebook --NotebookApp.max\_buffer\_size=99999999999999

Nous allons utiliser la bibliothèque ***Pytorch Geometric (PyG)*** dans ce TP. PyG est une bibliothèque basée sur ***PyTorch*** pour écrire et modéliser facilement des réseaux de neurones sur les graphes (GNN) pour un large éventail d'applications liées aux données structurées. Cette bibliothèque comporte aussi d’autres modèles d’apprentissage sur les graphes basés sur les marches aléatoires comme : Node2vec et DeepWalk.

Lien vers la documentation : <https://pytorch-geometric.readthedocs.io/en/latest/>

**Partie 1 (La classe Node2vec) :**

Lisez bien la documentation de la classe Node2vec du PyG, et répondez aux questions suivantes :

Lien vers la documentation : <https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html?highlight=Node2vec#torch_geometric.nn.models.Node2Vec>

* Quelle est la différence entre le paramètre **walk\_length** et **context\_size ?**
* Soit M une marche aléatoire , et supposant que **context\_size = 3,** combien d’échantillons positifs peut-on générer de cette marche ?
* À quoi sert le paramètre **num\_negative\_samples** ?
* Comment sont générées les marches négatives ?
* Que retourne le **loader** de Node2vec ?

**Partie 2 (Classification des nœuds) :**

Télécharger le notebook *TP2\_Node2vec\_Classification\_noeuds* et répondre aux questions.

1. <https://dl.acm.org/doi/10.1145/2939672.2939754> [↑](#footnote-ref-1)