Chapitre 5 : Les graphes : définition, opérations et usages

Progressivement mis en valeur depuis le XVIII ième siècle, les graphes sont des objets mathématiques très utiles pour modéliser des problèmes qui doivent prendre en compte un certain nombre d'objets mais aussi les relations qui les unissent. Leur étude a donné naissance à toute une branche des mathématiques qui s'attache à découvrir tous leurs secrets et propriétés. Mais, plus important, les graphes sont très utiles pour modéliser tout un ensemble de problèmes concrets (routes reliant des villes sur une carte, architecture réseau d'un ordinateur…) si bien que la création d'algorithmes d'optimisation à base de graphes n'est pas juste un exercice d'esprit mais amène à des progrès techniques. Le but de ce cours n'est pas de faire de vous des experts des graphes mais de vous présenter les propriétés principales de ces objets ainsi que les définitions à connaître pour pouvoir aborder la littérature scientifique les mentionnant. On présentera ensuite une série d'opérations et d'algorithmes très connus sur les graphes avant de voir quelles ont pu être leur utilisations principales en NLP.

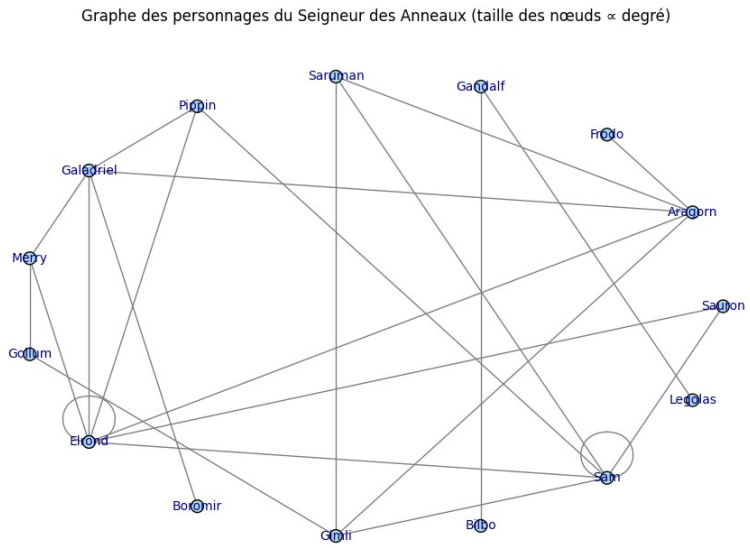
1) Définition mathématique d'un graphe

Un graphe est un objet mathématique qui représente des relations.
La théorie des graphes est une branche des mathématiques qui a commencé à être explorée au XVIII ième siècle et a connu un développement accéléré avec l'essor de l'informatique, les graphes ayant un vaste panel d'applications.L'objectif ici n'est pas de proposer un cours exhaustif sur les graphes — un sujet qui pourrait remplir un semestre entier — d'autant que de nombreuses ressources de qualité existent déjà[1]. Nous nous concentrerons plutôt sur les définitions essentielles des graphes et de leurs principales propriétés.

a) Définition d'un graphe et vocabulaire[1]

On définit un graphe comme un ensemble de nœuds (vertex, pluriel vertices) et d'arêtes (edges) connectant des nœuds.
Mathématiquement, on note G = (V,E) avec V l'ensemble des nœuds et E l'ensemble des arrêtes.
À priori un graphe est une structure mathématique indépendante de ce que représentent les nœuds et les arrêtes. Ainsi on peut dire que deux graphes sont égaux si ils ont le même nombre de nœuds et qu'il existe un mapping entre ces deux listes de nœud tel que s'il deux nœuds sont connectés par une arrête dans un graphe, alors leurs correspondants sont aussi connectés par une arrête. On parle aussi d'isomorphisme entre les graphes.

1 https://medium.com/basecs/a-gentle-introduction-to-graph-theory-77969829ead8 ressource bien illustrée



On dit que deux nœuds sont adjacents (adjacent) s'il existe au moins une arrête les reliant. On appelle le degré d'un nœud le nombre d'arêtes auxquelles un nœuds est connecté. Ainsi, dans le graphe ci dessus, le nœud « Galadriel » a un degré de 5. Un résultat important de la théorie des graphes (Euler) est dit lemme de la poignée de main : dans un graphe, la somme du degré de tous les nœuds est un nombre pair.

Un graphe peut potentiellement avoir plusieurs arrêtes partant et arrivant des mêmes nœuds (multipled edges, multigraph) mais c'est peu fréquent dans la pratique.

On appelle une boucle (loop) une arrête partant d'un nœud et arrivant sur le même nœud (« Sam »).

Un **chemin** dans un graphe est une séquence de nœuds $P=(v1,v2,\ldots,vk)$ telle que $\forall i \in [1,k-1],(vi,vi+1) \in E$. En d'autres termes, chaque paire consécutive de nœuds dans le chemin est connectée par au moins une arête.

Dans notre graphe, un chemin plus long pourrait être P=(Frodo,Sam,Gandalf,Aragorn,Legolas). Cela signifie que Frodo et Sam sont connectés, Sam est connecté à Gandalf, Gandalf à Aragorn, et Aragorn à Legolas. Visuellement, ce chemin peut être représenté par une série d'arêtes reliant les nœuds en séquence.

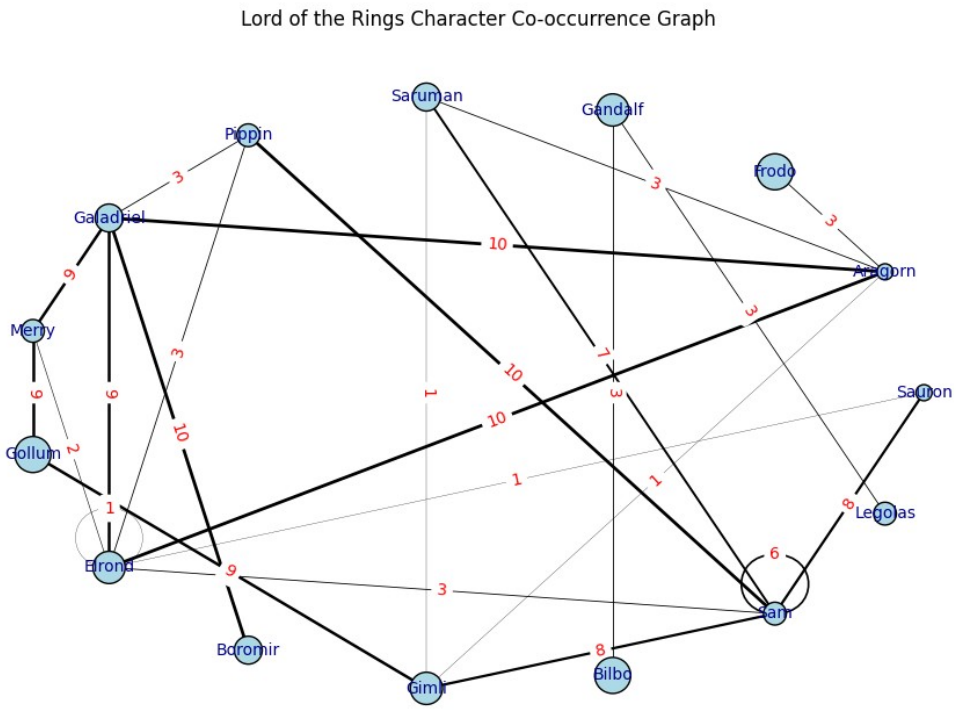
Un **cycle** est un chemin C=(v1,v2,…,vk,v1) qui commence et finit au même nœud sans répéter d'autres nœuds ni arêtes, sauf le premier et le dernier. Par exemple, si (Frodo,Sam),(Sam,Gandalf), (Gandalf,Frodo) $\in E$, alors C=(Frodo,Sam,Gandalf,Frodo) est un cycle.

Les cycles peuvent poser des problèmes pour plusieurs raisons. Premièrement, lors d'un parcours du graphe, un cycle peut entraîner un bouclage infini. Par exemple, si un algorithme d'exploration (comme une recherche en profondeur) traverse le cycle C sans mémoriser les nœuds visités, il peut naviguer indéfiniment entre Frodo, Sam et Gandalf. Deuxièmement, les cycles créent des dépendances circulaires dans des systèmes complexes. Par exemple, dans un graphe de dépendances de tâches, un cycle $A \rightarrow B \rightarrow C \rightarrow A$ signifie qu'aucune des tâches ne peut commencer sans que les autres soient terminées, ce qui est logiquement impossible. Enfin, les cycles peuvent rendre difficile l'interprétation des données. Dans un graphe de relations, un cycle peut introduire des ambiguïtés ou des redondances. Par exemple, si A est connecté à B, B à C, et C à A, il devient complexe d'interpréter la relation globale entre A, B, et C.

Un graphe dans lequel tous les nœuds sont connectés à tous les autres (c'est à dire que pour tout ei, ej dans E il existe un chemin dont ei est le premier nœud et ej le dernier) est dit connecté (connected).

On appelle un graphe planaire (planar) un graphe qui peut être dessiné sans que deux arrêtes ne s'intersectent.

On fait la distinction entre les graphes orientés (directed), pour lesquels la direction des arrêtes est importante et les graphes non orientés (undirected) pour laquelle elle ne l'est pas.
Pour un graphe non orienté, (Frodo, Sam) est équivalent à (Sam, Frodo), mais pas dans un graphe orienté. On note l'orientation des arrêtes par une petite flèche.



On distingue de même les graphes où toutes les arrêtes ont la même valeur (unweighted) des graphes dont les nœuds ont des poids numériques différents et que l'on appelle des graphes pondérés (weighted graphs). Ainsi dans le graphe précédent, une arrête représente le nombre de cooccurrences remarquées dans une phrase entre deux personnages. Le graphe est non orienté car la relation de « coocurrence dans un texte » est commutative.

Comme vous vous en doutez, les graphes pondérés, non pondérés, dirigés ou non ont des propriétés mathématiques et algorithmiques communes , mais d'autres différentes selon leurs spécificité. La plupart du temps, on travaille en informatique et en mathématiques avec des graphes orientés et acycliques (DAG, Directed Acyclic Graph) mais la plupart des graphes applicatifs contiennent des cycles.

Parmi les types de graphes particuliers on peut citer les graphes monopartites dans lesquels tous les nœuds sont de même nature et peuvent être reliés entre eux des autres, notamment les graphes bipartites où les nœuds peuvent être scindés entre deux groupes différent, et chaque connexion implique forcément deux nœuds de catégorie différente.

L'essentiel est de ne pas oublier que les graphes sont des outils de modélisation. Il faut donc choisir la représentation qui correspond au cas d'usage.

b) Différentes représentations possibles d’un graphe

Dans l'étude des graphes, il existe plusieurs façons de les représenter, chacune ayant des avantages en fonction des applications et des algorithmes à utiliser. La notation mathématique traditionnelle pour représenter un graphe est souvent sous la forme d'un couple G=(V,E), où V représente l'ensemble des sommets du graphe et E l'ensemble des arêtes, qui sont des paires de sommets reliés. Pour un graphe orienté, les arêtes sont alors des couples ordonnés de sommets, tandis que pour un graphe non orienté, les arêtes sont simplement des ensembles non ordonnés de deux sommets.

En informatique, les graphes sont souvent représentés sous forme de structures de données, telles que les listes d'adjacence ou les tableaux d'adjacence. Dans une liste d'adjacence, chaque sommet est associé à une liste contenant ses voisins directs. Cette représentation est généralement plus efficace en termes de mémoire lorsque le graphe est peu dense. En revanche, les tableaux d'adjacence utilisent un tableau pour chaque sommet, indiquant directement la connexion à d'autres sommets, souvent avec des valeurs binaires (1 pour connecté, 0 pour non connecté). Comme d’habitude l’informatique est très souple et offre de très nombreux moyens de représenter les graphes. Dans un programme on peut aussi représenter un graphe par :

* une liste d’arêtes (tuples)

* une liste d’adjacence (Un dictionnaire de voisins) : Dans cette approche, chaque sommet du graphe est associé à un dictionnaire, dont les clés sont les sommets voisins et les valeurs sont souvent des poids d'arêtes. Cela permet une gestion très flexible des graphes, en particulier dans les cas où les arêtes sont pondérées ou lorsque l’on a besoin de stocker des informations supplémentaires avec chaque arête

* En programmation orientée objet, une classe à part avec ses objets et méthodes à définir.

Informatiquement, notre graphe peut être représenté comme une liste de tuples que l’on appelle liste d’arrête:

edges = [("Aragorn", "Frodo"), ("Aragorn", "Gandalf"), ("Aragorn", "Gimli"), ("Aragorn", "Legolas"), ("Aragorn", "Sam"), ("Aragorn", "Sauron"), ("Aragorn", "Saruman"), ("Bilbo", "Gimli"), ("Boromir", "Elrond"), ("Elrond", "Galadriel"), ("Elrond", "Gollum"), ("Elrond", "Gimli"), ("Frodo", "Gandalf"), ("Galadriel", "Gollum"), ("Galadriel", "Merry"), ("Galadriel", "Pippin"), ("Gandalf", "Saruman"), ("Gimli", "Boromir"), ("Legolas", "Sam"), ("Merry", "Gollum"), ("Pippin", "Galadriel"), ("Sauron", "Legolas"), ("Sam", "Gimli")]

Cette représentation est la plus simple à concevoir, et est idéale quand le graphe comporte peu d’arrêtes.

Pour un graphe pondéré, c’est un triplet incluant le poids comme troisième élément.

On peut aussi représenter le graphe comme une liste d’adjacence. Il s’agit en fait d’un dictionnaire associant un nœud à la liste de noeux auxquels il est connecté.

graph = { "Aragorn": ["Frodo", "Gandalf", "Legolas", "Sam", "Saruman"], "Frodo": ["Aragorn"], "Gandalf": ["Aragorn", "Saruman", "Pippin"], "Legolas": ["Aragorn", "Sauron"], "Sam": ["Aragorn", "Gimli", "Bilbo"], "Saruman": ["Aragorn", "Gandalf"], "Pippin": ["Gandalf", "Galadriel"], "Galadriel": ["Pippin", "Merry", "Elrond"], "Merry": ["Galadriel", "Gollum"], "Gollum": ["Merry", "Elrond"], "Elrond": ["Galadriel", "Gollum", "Boromir", "Gimli"], "Boromir": ["Elrond", "Gimli"], "Gimli": ["Elrond", "Boromir", "Sam", "Bilbo"], "Bilbo": ["Gimli", "Sam"], "Sauron": ["Legolas"] }

Du côté des représentations matricielles, deux structures sont couramment utilisées : la matrice d'adjacence et la matrice d'incidence. La matrice d'adjacence est une matrice carrée dans laquelle chaque élément aij indique si les sommets i et j sont connectés par une arête. Si le graphe est orienté, la matrice d'adjacence sera asymétrique, tandis que dans le cas d'un graphe non orienté, elle sera symétrique. Quant à la matrice d'incidence, elle associe chaque arête à un ensemble de sommets. Dans cette représentation, chaque ligne correspond à une arête et chaque colonne à un sommet ; un élément de la matrice est généralement égal à 1 si l'arête correspondante est incidente au sommet, et 0 sinon. Pour un graphe pondéré, on remplace 0 par le poids de l'arrête. Cette matrice est particulièrement utile pour représenter des graphes avec de nombreuses arêtes mais relativement peu de sommets.

	Aragorn	Bilbo	Boromir	Elrond	Frodo	Galadriel	Gandalf	Gimli	Gollum	Legolas	Merry	Pippin	Sam	Sauron	Saruman
Aragorn	0	0	0	0	1	0	1	1	0	1	0	0	1	1	1
Bilbo	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Boromir	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
Elrond	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0
Frodo	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Galadriel	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0
Gandalf	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1
Gimli	1	1	1	1	0	0	0	0	0	0	0	0	1	0	0
Gollum	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0
Legolas	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
Merry	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0
Pippin	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Sam	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0
Sauron	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Saruman	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0

On peut aussi représenter un graphe par une matrice d'incidence qui indique quels liens arrivent sur quel sommet.

Une matrice d'incidence I d'un graphe non orienté avec n sommets et m arêtes est une matrice de taille n×m où :

- Chaque ligne correspond à un sommet.
- Chaque colonne correspond à une arête.
- Si une arête ej relie un sommet vi à un sommet vk, alors :
    - I[i,j]=1 si le sommet vi est incident à l'arête ej.
    - I[k,j]=1 si le sommet vk est incident à l'arête ej.
    - Toutes les autres entrées de cette colonne sont nulles.

Pour un graphe orienté, on mettrait 1 pour le sommet de départ et −1 pour le sommet d'arrivée.

Matrice d'incidence

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	e_{14}	e_{15}	e_{16}	e_{17}	e_{18}	e_{19}	e_{20}	e_{21}	e_{22}
Aragorn	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bilbo	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Boromir	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Elrond	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0
Frodo	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Galadriel	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0
Gandalf	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
Gimli	0	0	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	1	0	0	0
Gollum	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
Legolas	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
Merry	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Pippin	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Sam	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
Sauron	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Saruman	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Toutes ses représentations différentes décrivent en réalité le même objet mathématique, sans la moindre différence. On peut parler de représentations isomorphiques. Néanmoins ces représentations ont des propriétés computationnelles différentes. Il est selon les opérations que l'on veut faire (traversée, calcul de flots) et la topologie du graphe (beaucoup de nœuds mais pas d'arrêtes, beaucoup d'arrêtes pour peu de nœuds…) plus souhaitable de favoriser l'une ou l'autre de ces représentations. En tout cas il faut toutes les connaître :)

Représentation	Espace $O(\cdot)$	Vérifier (u, v) existe ?	Trouver voisins de u	Recommandé pour…
Matrice d'adjacence	$O(n^2)$	$O(1)$	$O(n)$	Graphes denses
Liste d'adjacence	$O(n+m)$	$O(\deg(u))$	$O(\deg(u))$	Graphes clairsemés
Liste d'arêtes	$O(m)$	$O(m)$	$O(m)$	Stockage compact
Matrice d'incidence	$O(n \times m)$	$O(m)$	$O(n)$	Théorie des graphes

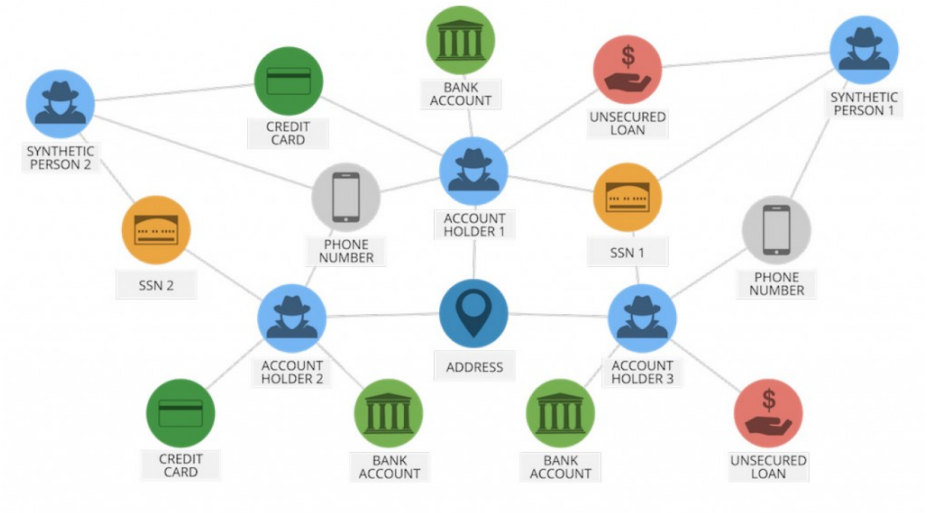
c) Pour aller plus loin : nouveaux types de graphe

Les graphes mathématiques décrivent avant tout des structures. On étudie la connexion entre des objets et ce que l'on peut déduire ce ces connexions, indépendamment de la nature des objets reliés et indépendamment de la nature des relations qui relient ces objets. Or dans la réalité, les graphes que l'on utilise pour modéliser des situations ont une sémantique plus riche et plus complexe. Notamment les nœuds représentent des objets avec des propriétés, les relations sont définies par plus que simplement une direction et leur poids… Les graphes en tant qu'objet mathématique ont donc une expressivité limitée pour décrire la complexité de la réalité. Des tentatives ont donc été proposer pour modéliser des situations et résoudre des problèmes plus complexes avec des graphes.

Ces tentatives ont donné naissance à des extensions et à des variations des graphes classiques, permettant de mieux capturer la richesse et la complexité des situations réelles. Par exemple, les graphes *étiquetés* ou *annotés* introduisent des propriétés supplémentaires pour les sommets et les arêtes. Ainsi, chaque sommet peut être associé à un ensemble de caractéristiques, comme un nom, une position géographique, ou des attributs qualitatifs, tandis que les arêtes peuvent être étiquetées avec des poids, des types de relations ou même des valeurs temporelles, ce qui permet de modéliser des phénomènes tels que les distances, les capacités de communication, ou les relations temporelles.

En parallèle, des concepts comme les *graphes multiorientés* et les *graphes pondérés* ont émergé pour capturer des relations plus complexes. Les graphes multiorientés, par exemple, permettent de modéliser des situations où plusieurs types de relations différentes existent entre deux objets. Dans un réseau social, par exemple, on pourrait avoir des arêtes représentant des amitiés, des relations professionnelles ou des interactions publiques, chacune ayant une signification distincte et nécessitant une représentation séparée. De même, les graphes pondérés vont au-delà de la simple connexion binaire entre deux nœuds, en attribuant des valeurs numériques aux arêtes, représentant par exemple la force, la durée ou le coût d'une relation.

Et, il faut s'y faire, c'est la plupart du temps ces graphes complexes qui sont utilisés dans des applications réelles. Voir cet exemple sur la lutte contre la fraude[2] qui présente plusieurs types de nœuds appartenant à différentes classes. Les nœuds peuvent être complétés par des attributs (pour un client sa date de naissance, son numéro bancaire…). Les relations aussi peuvent être chargées d'attribut : si une banque fait un prêt : de combien d'argent ? Aussi les nœuds peuvent être reliés par des arrêtes de type différents. Un client peut posséder un compte dans une banque, faire un virement depuis ce compte, demander sa fermeture… Le graphe doit être capable de refléter toute cette diversité de situations.

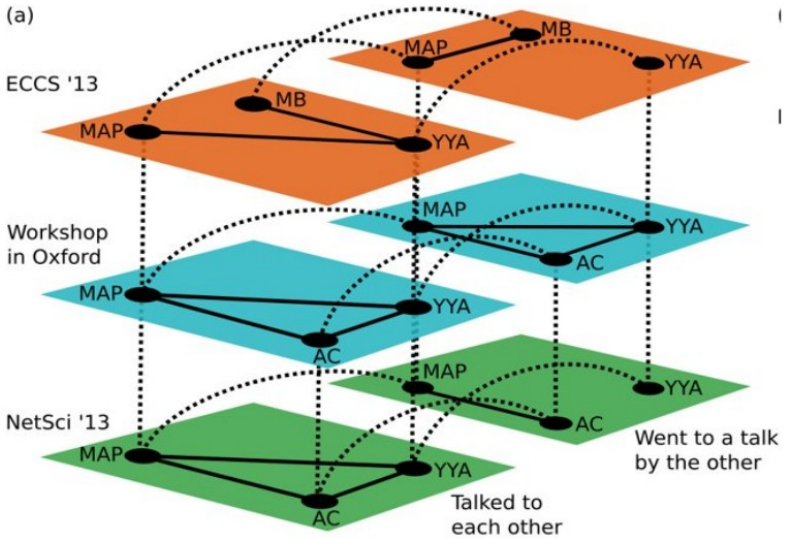


Par ailleurs, les graphes peuvent également être enrichis par des *attributs dynamiques*, où les propriétés des sommets et des arêtes évoluent dans le temps. Cela permet de modéliser des systèmes en constante évolution, comme les réseaux de transport ou les systèmes sociaux, où les connexions et les objets peuvent changer à chaque instant. De telles représentations sont particulièrement utiles pour résoudre des problèmes impliquant des dynamiques complexes, tels que la propagation de l'information ou l'analyse de l'évolution des structures sociales.

Un *multiplex graph* (ou graphe multiplex) est un type de graphe dans lequel plusieurs types de relations ou d'interactions entre les mêmes ensembles de nœuds (sommets) coexistent simultanément. Chaque type de relation est représenté par une arête distincte, mais les nœuds sont les mêmes dans tous les types de relations. Autrement dit, un graphe multiplex est une extension des graphes classiques dans lesquels plusieurs couches (ou types de relations) sont superposées pour capturer la complexité des interactions entre les objets.Dans un graphe multiplex, chaque couche représente une relation différente entre les nœuds. Ce type de graphe peut par exemple représenter plus finement les modes de transport entre des villes : certaines villes disposent d'un aéroport, d'un port, d'une gare, d'autoroutes… soit différents moyens de transport. On peut donc dessiner un graphe pondéré (en temps/coût de trajet) où chaque couche du graphe représente un mode de locomotion particulier.

2 https://neo4j.com/blog/fraud-detection/enterprise-fraud-detection/

Ainsi, les nœuds des graphes finissent par avoir un certain nombre de propriétés typées ou non typées qui les rapprochent d'une ligne d'une table dans une base de donnée relationnelle, avec des propriétés et des attributs. Tous les nœuds n'ont pas forcément de valeurs pour chacun des attributs.



Une autre façon de considérer cet objet serait de considérer le graphe total comme une série de graphes parallèles. Chaque graphe (un plan horizontal sur l'illustration) correspond à un champ particulier du graphe total, et des axes verticaux relient tous les attributs relevant du même nœud. Si ces nouveaux objets mathématiques permettent de décrire des phénomènes plus complexes, ils sont eux aussi bien plus complexes à manier conceptuellement comme informatiquement. Des travaux sont encore en cours pour déterminer des méthodes de clustering et d'embedding pour ces objets.

2) Les propriétés essentielles des graphes

Les graphes sont des structures mathématiques qui modélisent des objets et leurs relations. Ils sont définis par des sommets (objets) et des arêtes (relations), avec des propriétés essentielles telles que la **connectivité** (relation entre les nœuds), le **degré** (nombre de connexions d'un sommet), et la **cyclicité** (présence ou absence de cycles). Les graphes peuvent être **pondérés** (avec des valeurs sur les arêtes) et **orientés** (relations directionnelles). Ces propriétés permettent d'analyser des systèmes complexes, d'identifier des chemins optimaux, de détecter des cycles, d'étudier la robustesse d'un réseau et de résoudre des problèmes d'optimisation comme le plus court chemin ou le flux maximum.

1) chemins et traversées

L'avantage de modéliser ses données sous forme de graphe est que cela permet de voir immédiatement si des points sont connectés ou non. Néanmoins sur des graphes de grande taille, cette information est insuffisante. En effet dans un graphe connecté, tout point est accessible depuis un autre. On est donc intéressé par le plus court chemin entre deux nœuds. Différents algorithmes ont été inventés pour résoudre le problème. Ils font partie des « grands classiques » du cours

d’algorithmique, sont au programme de toute licence d’informatique en France et on peut demander de les appliquer dans des entretiens. L’idée est déjà de savoir expliquer leur fonctionnement en langage naturel, quelle est la différence entre chacun des algorithmes, dans quel cas il vaut mieux utiliser l’un plutôt que l’autre. Les réimplémenter en python est simple et pourrait être un bon exercice.[3]

DFS (Depth-First Search) : Parcours un graphe de manière récursive, utilisé pour explorer la structure d’un graphe, détecter des cycles ou trouver des composantes connexes. Utile pour les arbres de décision et la recherche de cycles.

BFS (Breadth-First Search) : Parcours un graphe de manière itérative, utilisé pour trouver le plus court chemin dans un graphe non pondéré. Utilisé dans les réseaux sociaux et la recherche de chemins dans des graphes de connexion.

Dijkstra : Calcule le plus court chemin à partir d’un sommet source vers tous les autres sommets dans un graphe pondéré avec des arêtes positives. Utilisé dans les systèmes de navigation, les réseaux de transport et les recommandations.

Bellman-Ford : Similaire à Dijkstra, mais gère les arêtes de poids négatifs et détecte les cycles de poids négatif. Utilisé dans les réseaux financiers et l’analyse des arbitrages.

Floyd-Warshall : Trouve les plus courts chemins entre toutes les paires de sommets dans un graphe, adapté aux graphes avec des poids négatifs (sans cycles négatifs). Utilisé dans les réseaux de communication et le calcul de distances entre tous les nœuds.

A* : Extension de Dijkstra utilisant une heuristique pour guider la recherche du plus court chemin. Utilisé en robotique, jeux vidéo et navigation GPS.

Johnson : Combine Bellman-Ford et Dijkstra pour calculer les plus courts chemins entre toutes les paires de sommets dans un graphe avec des poids négatifs. Utilisé pour des graphes complexes avec poids négatifs.

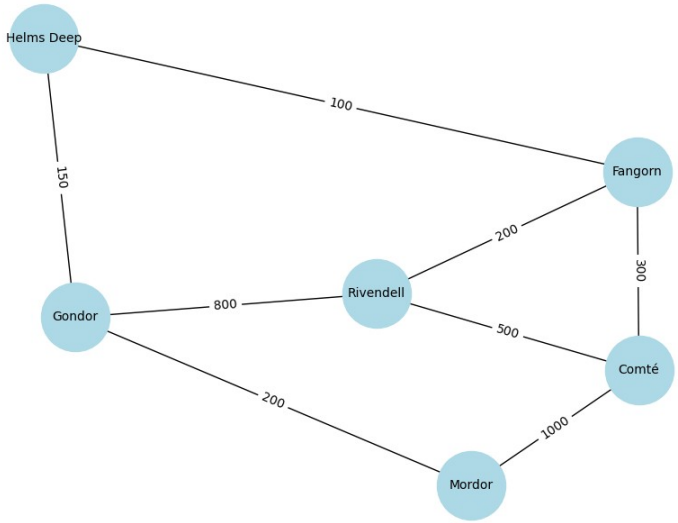
Bidirectional Search : Recherche simultanée depuis le sommet source et le sommet destination. Plus efficace que la recherche unidirectionnelle pour les grands graphes. Utilisé dans les systèmes de navigation et les graphes sociaux.

Un exemple : l’algorithme de Djikstra

L'algorithme de Dijkstra permet de trouver le plus court chemin entre un nœud de départ et tous les autres nœuds d'un graphe. Cela est particulièrement utile pour des graphes représentant des réseaux, des cartes géographiques, etc. C’est un grand classique et la plus grande fierté de la nation néerlandaise.

L’algorithme permet de trouver le plus court chemin d’un nœud vers tous les autres dans un graphe pondéré dont tous les poids sont POSITIFS (risque de passer à l’infini par une arrête négative sinon). L’idée est que l’on calcule les distances du point de départ vers tous les autres (initialisées à l’infini) et on utilise une file pour extraire les arrêtes qui ont le poids le plus faible pour mettre à jour la distance à parcourir depuis le point de départ (relâche).

3 https://medium.com/pythoneers/graph-theory-algorithms-from-mathematical-concepts-to-python-code-93427e86f78c




```python
graph = {
    'Comté': {'Rivendell': 500, 'Mordor': 1000, 'Fangorn': 300},
    'Rivendell': {'Comté': 500, 'Gondor': 800, 'Fangorn': 200},
    'Gondor': {'Rivendell': 800, 'Mordor': 200, 'Helms Deep': 150},
    'Mordor': {'Comté': 1000, 'Gondor': 200},
    'Fangorn': {'Comté': 300, 'Rivendell': 200, 'Helms Deep': 100},
    'Helms Deep': {'Gondor': 150, 'Fangorn': 100}
}
```

```python
def dijkstra(graph, start):  1 usage
    # Distance des nœuds, initialisée à l'infini sauf pour le nœud de départ
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    # File de priorité pour explorer les nœuds
    priority_queue = [(0, start)]

    while priority_queue:
        # Extraction du nœud avec la distance minimale
        current_distance, current_node = heapq.heappop(priority_queue)

        # Si cette distance est déjà plus grande que celle enregistrée, on ne fait rien
        if current_distance > distances[current_node]:
            continue

        # Exploration des voisins
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight
            # Si on trouve un chemin plus court on met à jour (relâche) les poids
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush( *args: priority_queue, (distance, neighbor))

    return distances
```

Quelle est la complexité de cet algorithme ?

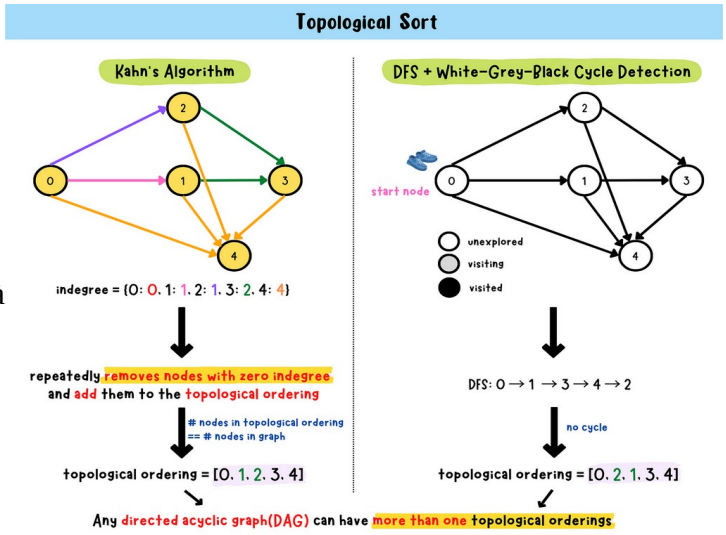
2) questions d’optimisation

Représenter un problème avec des graphes permet également de résoudre d’autres types de problèmes avec des techniques mathématiques. L’objet et le temps de ce cours ne permettent pas de tout explorer. Mais il faut savoir que des solutions et algorithmes existent pour ces problèmes. Les graphes offrent une large gamme de problèmes d'optimisation en plus du calcul des plus courts chemins, tels que le coloriage de graphes, les flux et la planification.

Coloriage de graphes : L'objectif est d'assigner des couleurs à chaque sommet d’un graphe de manière à ce que deux sommets adjacents n'aient jamais la même couleur. Ce problème est utilisé dans la planification d'horaires, les problèmes de fréquence radio, et la gestion des ressources dans les réseaux.

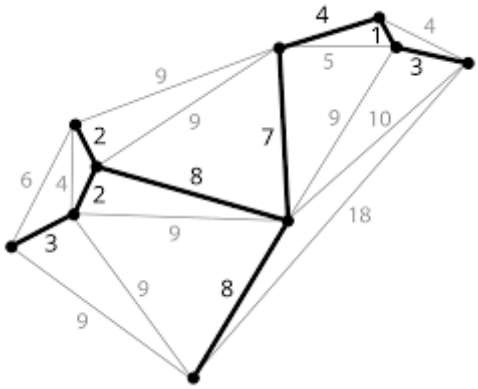
Le tri topologique : Le tri topologique est un algorithme permettant d'ordonner les sommets d'un graphe orienté acyclique en respectant les relations de dépendance entre eux. Son objectif est de trouver un ordre linéaire tel que si un sommet A précède un sommet B dans le graphe, alors A apparaîtra toujours avant B dans la liste finale. Cette méthode est particulièrement utile pour organiser des tâches devant être exécutées dans un ordre précis, comme dans la planification de projets, la compilation de fichiers source ou l'ordonnancement d'instructions dans un pipeline informatique.

Il existe deux approches principales[4] pour effectuer un tri topologique. La première est l'algorithme de Kahn, qui repose sur l'élimination progressive des sommets sans prédécesseurs. Dans un premier temps, on identifie tous les sommets du graphe qui n'ont pas d'arête entrante et on les ajoute à une liste ordonnée. Ensuite, on supprime ces sommets du graphe ainsi que leurs arêtes sortantes, ce qui permet de révéler de nouveaux sommets sans prédécesseurs. Ce processus est répété jusqu'à ce que tous les sommets soient traités. La seconde approche utilise un parcours en profondeur. On explore chaque sommet en suivant ses arêtes sortantes et, une fois qu'un sommet n'a plus de voisins à visiter, on l'ajoute en tête d'une liste. Après avoir parcouru l'ensemble du graphe, la liste obtenue représente l'ordre topologique des sommets.



Problèmes de flux : Les problèmes de flux concernent l'acheminement de ressources (comme des marchandises ou des données) à travers un réseau, tout en respectant certaines contraintes de capacité. Le problème de flux maximum, par exemple, vise à maximiser le flux d'une source vers une destination dans un graphe en tenant compte des capacités des arêtes. Ce type de problème est crucial dans les réseaux de transport, les systèmes de communication et l'allocation de ressources. (algorithme de Ford-Fulkerson)

Couplages et arbres couvrants : Les problèmes de couplage concernent l'association de sommets dans un graphe de manière optimale, souvent en maximisant le nombre de paires. Les arbres couvrants, quant à eux, visent à trouver un sous-ensemble d'arêtes qui relient tous les sommets sans former de cycles. Ces problèmes sont utilisés dans les réseaux de télécommunications et les arbres de décision. L'usage d'arbres couvrants minimaux est utile pour la planification de tâche et pourrait servir dans le développement de règles dans une ontologie portant sur des processus industriels par exemple. (algorithmes de Prim ou Kruskal)



4 https://yuminlee2.medium.com/topological-sort-cf9f8e43af6a

3) Analyse en réseau (network / community analysis)

La théorie des réseaux est une science au carrefour entre plusieurs disciplines, dont des sciences humaines (archéologie, biologie, économie, sociologie...). En mathématiques / informatique, elle est vue comme une partie de la théorie des graphes. On définit un réseau comme un graphe dont les nœuds et/ou les arrêtes ont des attributs et on analyse les réseaux selon les relations symétriques et asymétriques entre leurs composants. Une des application les plus connue est l'analyse des réseaux sociaux. Les entités sont souvent des personnes ou des organisations, sites, publications. L'application de l'analyse de réseaux repose sur plusieurs techniques mathématiques :

L'exploration des mesures de centralité permet d'analyser les graphes en identifiant les nœuds ou les arêtes les plus importants, souvent dans un contexte de réseau social, de transport ou d'information. Les méthodes mathématiques pour mesurer la centralité sont variées et comprennent notamment :

Calcul matriciel : Les valeurs propres des matrices d'adjacence, ainsi que les vecteurs propres associés, permettent d'évaluer la centralité d'un nœud dans un graphe en utilisant la décomposition matricielle. Par exemple, la centralité propre est une mesure qui prend en compte les connexions d'un nœud à d'autres nœuds importants du graphe, ce qui permet de repérer des nœuds particulièrement influents ou bien connectés.

Analyse des degrés et degrés de centralité : La centralité par le degré est l'une des formes les plus simples, qui consiste à compter le nombre de connexions (ou arêtes) qu'un nœud possède. Plus un nœud a de connexions, plus il est central dans le réseau. Cela est utile pour repérer des nœuds fortement connectés, tels que les hubs, qui sont des points de passage obligés pour beaucoup d'autres nœuds.

Définition de nouvelles métriques comme la modularité : La modularité est utilisée dans l'analyse de la structure communautaire d'un graphe. Elle mesure la qualité d'une division du graphe en sous-graphes (communautés), en maximisant les arêtes internes et minimisant les arêtes externes aux communautés. Un graphe fortement modulaire aura des communautés bien séparées et des hubs qui relient ces communautés.

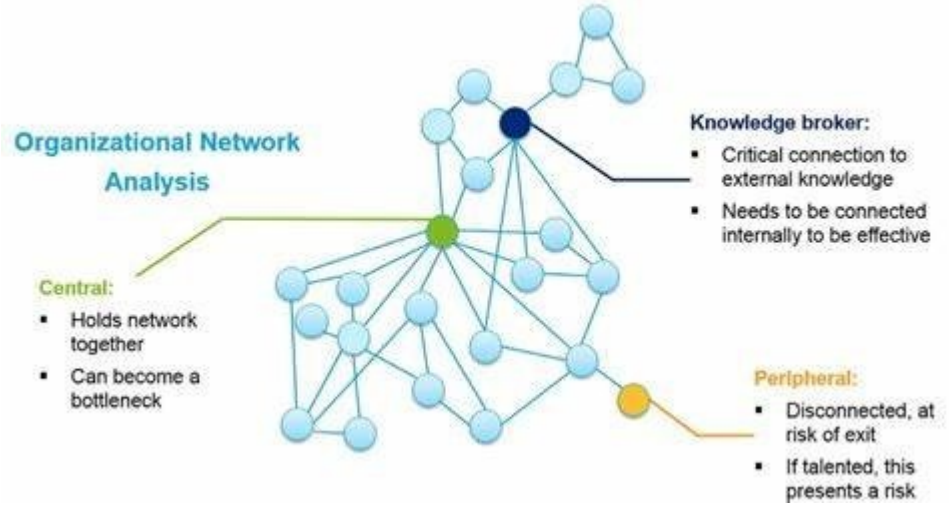
La formule de la modularité Q est la suivante :

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

où :

- A_{ij} est l'élément de la matrice d'adjacence, représentant l'existence d'une arête entre les sommets i et j (1 si une arête existe, 0 sinon),
- k_i et k_j sont les degrés des sommets i et j, soit le nombre total d'arêtes incidentes à ces sommets,
- m est le nombre total d'arêtes dans le graphe,
- $\frac{k_i k_j}{2m}$ représente la probabilité attendue qu'une arête existe entre i et j dans un graphe aléatoire conservant les degrés des sommets,
- $\delta(c_i, c_j)$ est une fonction indicatrice qui vaut 1 si les sommets i et j appartiennent à la même communauté et 0 sinon.

Ces mesures permettent de repérer des types de nœuds particuliers comme les **hubs** (nœuds par lesquels tout le monde passe pour aller vers d'autres nœuds), qui sont essentiels pour les applications pratiques telles que la propagation de l'information ou les systèmes de transport. La centralité peut aussi aider à identifier des **nœuds centraux** qui contrôlent l'accès à de nombreuses parties du réseau, des **nœuds d'autorité** dans des réseaux d'information, ou des **nœuds critiques** dont la suppression pourrait déstabiliser le réseau[5].



Ces méthodes permettent aussi de quantifier l'importance relative d'un nœud par rapport aux autres nœuds. Dans la théorie des graphes, les **hubs** sont des nœuds avec un grand nombre de connexions. Leur comportement de liaison peut être caractérisé par **l'assortativité** et **la disassortativité**.
Hubs assortatifs : préfèrent se connecter à d'autres hubs.
Hubs disassortatifs : évitent les autres hubs et privilégient les nœuds moins connectés.
Hubs neutres : se connectent aléatoirement, sans préférence pour les nœuds fortement ou faiblement connectés.

Une autre application des ces techniques consiste à repérer des récurrences structurelles dans l'organisation des réseaux. L'analyse de **patterns** dans les réseaux consiste à identifier des structures récurrentes ou des comportements spécifiques au sein du réseau. Cela permet de détecter des communautés, des motifs de connexion ou des tendances de propagation de l'information. Par exemple, l'analyse de **communautés** cherche à regrouper des nœuds qui sont plus densément connectés entre eux qu'avec le reste du réseau. L'identification de **motifs de sous-graphes** (ou **motifs fréquents**) permet de repérer des configurations récurrentes, comme des triangles ou des chemins spécifiques, qui révèlent des interactions ou des structures intéressantes. Ces techniques sont utilisées pour explorer des phénomènes comme la propagation des maladies dans un réseau social, les comportements d'achat dans un système de recommandation, ou la détection de fraudes dans des réseaux financiers. L'analyse de patterns aide à comprendre les dynamiques internes des réseaux complexes et à prédire leur évolution.

Un des défis encore à relever dans cette branche de la théorie des graphe est la prédiction de liens : dans un réseau, on peut avoir besoin de prédire des nouveaux lien (industrie pharmaceutiques, prédire de nouveaux risques et contre indications potentielles selon des patterns) et le macine learning y est de plus en plus lourdement impliqué. Un autre cas d'usage fréquent est la détection de fraude et de comportement anormaux (on parlerait plutôt de classification de nœuds ou de liens).

Ces techniques algorithmiques peuvent être assez délicates à comprendre au début, mais elles ont pour la plupart été implémentées dans des librairies. Vous pouvez donc assez facilement les exploiter pour faire des analyses de données intéressantes. C'est un générateur à posts Medium bien illustrés.[6]

5 https://www.linkedin.com/pulse/organizational-network-analysis-ona-andre-ribeiro-mcc/

3) Les graphes dans le domaine du traitement du langage naturel

Les graphes sont particulièrement utiles dans le domaine du traitement du langage naturel car ils permettent de modéliser de manière explicite et flexible les relations complexes entre différents éléments d'un texte, tels que les mots, les entités ou les concepts. En représentant ces relations sous forme de nœuds et d'arêtes, les graphes offrent une structure idéale pour résoudre de nombreuses tâches de traitement de l'information en langage naturel. Leur capacité à capturer des dépendances contextuelles et des interactions entre différents composants d'un texte les rend essentiels pour diverses applications.

a) Les bases de connaissance (Knowledge graph)

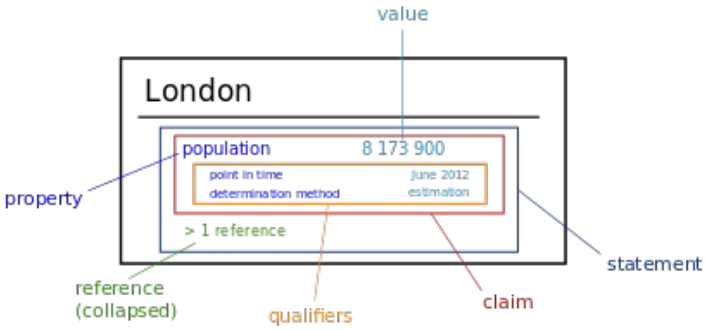
Les graphes sont utilisés de manière centrale pour modéliser des **bases de connaissances**

. Ces bases permettent de structurer l'information de manière à faciliter l'extraction, la compréhension et l'interprétation des données sémantiques contenues dans un texte.

Un des premiers et principaux projets dans ce domaine a été **WordNet**, lancé par Princeton en 1995. Il s'agit d'une base de données lexicale de l'anglais, qui regroupe les mots en ensembles de synonymes appelés **synsets**. WordNet structure les mots en fonction de relations sémantiques fondamentales telles que la **synonymie**, l'**hyperonymie** (relation de type "is_a"), la **meronymie** (partie-tout), et l'**antonymie**. Bien que son utilisation première ait été dans la recherche en linguistique computationnelle, WordNet est encore largement utilisé aujourd'hui pour la gestion des synonymes, la recherche d'information, et la disambiguïté sémantique.

DBpedia, lancé en 2007, représente une autre avancée importante dans l'utilisation des graphes en NLP. Cette base de connaissances extrait les données structurées de Wikipedia, notamment les informations présentes dans les infoboxes (les encadrés en haut à droite des articles). Avec plus de 2 milliards de triplets, DBpedia permet de relier des entités comme des personnes, des lieux ou des événements, facilitant ainsi des tâches comme la recherche d'information, l'extraction d'entités nommées (NER), et la construction de systèmes de question-réponse.

Un autre projet majeur est **WikiData**, lancé par D. Bollacker et al. en 2008. Contrairement à DBpedia, WikiData est une base de données multilingue, dynamique et évolutive qui contient une énorme quantité d'entités liées entre elles. Ce projet est unique en ce qu'il permet une gestion de données à grande échelle, offrant une ressource précieuse pour les applications NLP, notamment dans les domaines de la traduction automatique, de l'extraction d'informations et de la construction de représentations sémantiques.



Enfin, **Google Knowledge Graph** (KG)[7],
lancé en 2012, constitue un autre exemple de base de connaissances largement utilisée dans le

6 https://medium.com/web-mining-is688-spring-2021/twitter-networkx-graph-analysis-cef99c23fce9 voir par exemple cette analyse de données de Twitter à partir des outils déjà implémentés dans la librairie networkx.

7 https://support.google.com/knowledgepanel/answer/9787176?hl=FR

monde du NLP. Ce graphe relie des entités comme des personnes, des lieux, des objets, et des concepts, en utilisant des relations sémantiques pour enrichir les résultats de recherche de Google. Le Knowledge Graph permet d'améliorer la compréhension des requêtes des utilisateurs et de fournir des réponses plus contextuelles et précises. Les usages du Knowledge graph de Google sont divers : Le Knowledge Graph est une base de données sémantique qui relie des entités (personnes, lieux, concepts, événements, etc.) par des relations définies. Lorsqu'un utilisateur effectue une recherche, Google exploite ces informations pour enrichir les résultats de plusieurs façons :

Compréhension des requêtes : Grâce au KG, Google peut interpréter les intentions derrière les recherches, même lorsqu'elles sont formulées de manière ambiguë. Par exemple, si l'utilisateur recherche "Tesla", Google peut distinguer entre Nikola Tesla (l'inventeur) et Tesla (l'entreprise automobile) en fonction du contexte de la requête.

Fournir des réponses directes : Les résultats enrichis, comme les encarts affichant des faits (date de naissance, biographies, etc.), proviennent du Knowledge Graph. Ces encarts, appelés "knowledge panels", résument des informations pertinentes sur l'entité recherchée.

Suggestions de recherches connexes : En utilisant les relations entre les entités, Google propose des recherches complémentaires ou élargies. Par exemple, si vous recherchez un film, il peut suggérer des informations sur le réalisateur, les acteurs ou des films similaires.

Personnalisation des résultats : Google peut ajuster les résultats en fonction des préférences de l'utilisateur, en exploitant les relations dans le KG pour prioriser des informations localisées ou pertinentes pour une région spécifique.

Résolution des ambiguïtés : Lorsqu'une recherche contient un mot ou une expression pouvant désigner plusieurs entités (par exemple "Apple"), le KG permet de distinguer entre les différentes significations possibles en fonction du contexte.

Le Knowledge Graph joue donc un rôle clé dans la transition de Google vers un moteur de recherche "sémantique", capable de comprendre les relations et les significations au-delà de simples mots-clés.

Ce n’est pas la première fois que Google se sert de graphes pour optimiser les résultats de son moteur de recherche. En effet, un des algorithmes fondamentaux du moteur est également à base de graphes. Il s’agit de l’algorithme PageRank (Larry Page and Sergey Brin et nommé d’après son auteur, d’ailleurs, pas depuis l’idée de « page » web) qui a été un acteur clé du succès de l’entreprise. Alors qu’avant les moteurs de recherche reposaient surtout sur les mots clés uniquement, Page Rank propose une intuition différente : les pages webs les plus importantes sont aussi celles qui sont citées le plus souvent, et surtout celles qui sont citées par les sites les plus fiables. Le poids d’une page est donc conditionné par une forme de « vote de confiance » lié au nombre de pages qui pointent vers elle (parallèle avec le monde académique où un bon papier est un papier beaucoup cité).

On calcule le poids d’une page à partir de la formule mathématique suivante :

$$PR(p_i) = (1 - d) + d \cdot \sum_{p_j \in L(p_i)} \frac{PR(p_j)}{C(p_j)}$$

PR(pi) : Le score PageRank de la page pi. C'est la mesure de l'importance de la page pi dans le graphe et que l’on souhaite calculer.

d : Le facteur de **damping** (ou d'atténuation), généralement choisi entre 0.85 et 0.90. Il modélise la probabilité qu'un utilisateur continue de suivre des liens sur une page ou décide de visiter une autre page de manière aléatoire.
L(pi) : L'ensemble des pages pj qui contiennent un lien vers pi (les pages qui pointent vers pi).

PR(pj) : Le score PageRank de la page pj, qui pointe vers pi.
C(pj) : Le nombre total de liens sortants depuis la page pj. Le PageRank est divisé également entre toutes les pages vers lesquelles pj pointe (une page qui cite tout et n'importe quoi donne moins d'importance au fait de renvoyer vers un lien).

Ce calcul du score d'une page web est rendu possible par le fait que l'on représente l'ensemble des pages web comme un graphe orienté géant où les nœuds sont les pages web et les arrêtes les liens hypertextes menant d'une page pi à une page pj. En d'autres termes C(pj) correspond au cardinal sortant de pj.

On calcule donc le score d'une page en fonction du score d'une autre. Les poids sont donc « entraînés »[8]. Au départ le poids de toutes les pages est fixé à la même valeur. Puis on met à jour le score de toutes les pages une à une (en triant les pages au hasard). À la fin de cettre première itération, chaque page a maintenant un poids différent, mais celui de la première page mise à jour n'est pas représentatif, il a été calculé avec des poids initiaux. On réitère donc le processus plusieurs fois jusqu'à ce que les scores se stabilisent.

Si la version « vanilla » est facile à implémenter[9], Google a du optimiser son calcul pour des millions de page et penser un processus de mise à jour régulière du score des pages. Il est amusant de voir que si cette approche par Page Rank qui ignore toute notion de sémantique est diamétralement à l'opposé des travaux plus récents de l'entreprise sur la recherche augmentée via des bases de connaissances, les deux approches sont toutes deux appuyées sur les graphes.

b) Des graphes pour modéliser le language

Les graphes jouent un rôle essentiel dans la modélisation du langage, offrant une représentation structurée des relations entre les unités linguistiques. Ils ont toujours été utilisés pour représenter des processus stochastique. Claude Shannon, pionnier de la théorie de l'information, a mené des expériences sur la prédictibilité des syllabes, démontrant que le langage peut être modélisé comme un processus stochastique. Ces travaux ont inspiré des modèles probabilistes où les transitions entre unités linguistiques sont représentées par des graphes, chaque arête indiquant la probabilité de passage d'une unité à une autre et on peut représenter son expérience comme un graphe de transition entre les syllabes. Les **automates finis**, par exemple, sont des graphes utilisés pour modéliser des séquences de symboles, facilitant ainsi l'analyse syntaxique et la reconnaissance de motifs linguistiques.

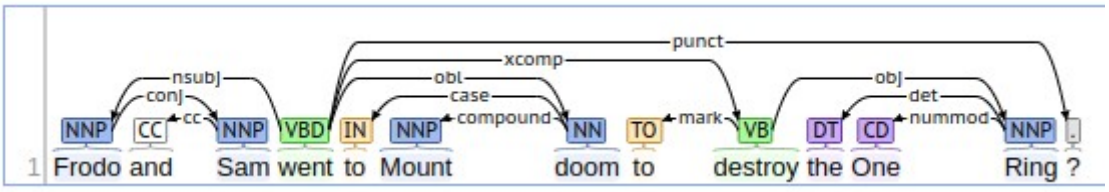
Les LLM fonctionnent de manière similaire, prédisant le prochain token dans une séquence en se basant sur les tokens précédents. Ce processus peut être visualisé comme un parcours dans un graphe probabiliste, où chaque nœud représente un état du contexte linguistique, et les arêtes sont pondérées par les probabilités de génération des tokens suivants. Ainsi, les LLMs peuvent être considérés comme des systèmes stochastiques naviguant dans un espace de possibilités linguistiques modélisé par un graphe. Cette représentation graphique permet de mieux comprendre le fonctionnement interne des LLMs et leur capacité à générer du texte cohérent en capturant les dépendances contextuelles et les structures linguistiques complexes.

Dans le domaine de la syntaxe, la théorie linguistique de Noam Chomsky a révolutionné la compréhension de la syntaxe en introduisant le concept de **grammaire générative et transformationnelle**. Cette approche modélise la structure des phrases à l'aide d'**arbres syntaxiques**, où chaque nœud représente une catégorie grammaticale et les branches illustrent les

8 Il ne s'agit pas d'un entraînement au sens ML du terme, car il n'y a pas de paramètres qui sont appris, mais plutôt de la convergence d'un processus stochastique. On en reparlera au chapitre 6 avec l'exemple de kmeans.
9 https://ithinkbot.com/the-pagerank-algorithm-at-the-core-of-googles-search-dominance-76767f718ab8

relations hiérarchiques entre ces catégories. Ces arbres permettent de visualiser la manière dont les constituants d'une phrase se combinent pour former des structures syntaxiques valides. La grammaire et ses règles est elle même régie par des notions sur les graphes. Ainsi certaines variantes de la syntaxe générativiste imposent que le graphe soit planaire (les arrêtes ne se recouvrent pas. Ce qui implique que les liens de dépendance soient unidirectionnels dans le sens de la lecture).

En traitement automatique du langage naturel (NLP), l'**analyse syntaxique** consiste à construire de tels arbres à partir de phrases données. Ce processus, appelé **parsing**, identifie la structure grammaticale en déterminant les relations entre les mots et les groupes de mots. L'analyse syntaxique est essentielle pour des applications telles que la traduction automatique, la reconnaissance d'entités nommées et la compréhension du langage naturel.



Les modèles d'apprentissage automatique jouent un rôle crucial dans l'automatisation de l'analyse syntaxique. Le **parsing** est abordé comme une tâche de prédiction et de classification des liens entre les constituants d'une phrase. Des algorithmes tels que les **réseaux neuronaux** et les **modèles probabilistes** sont entraînés sur de vastes corpus annotés pour apprendre à prédire les structures syntaxiques correctes. Ces modèles peuvent ensuite généraliser à de nouvelles phrases, facilitant ainsi l'analyse syntaxique automatique à grande échelle. Les **réseaux de neurones graphiques** (Graph Neural Networks - GNNs) ont récemment montré leur efficacité dans cette tâche. Ils permettent de capturer les dépendances complexes entre les mots en modélisant les phrases sous forme de graphes, où les nœuds représentent les mots et les arêtes les relations potentielles entre eux. Cette approche offre une flexibilité accrue par rapport aux méthodes traditionnelles basées sur des arbres stricts. Par exemple, l'article "Strongly Incremental Constituency Parsing with Graph Neural Networks" propose un système de transition appelé "attach-juxtapose" qui construit de manière incrémentale un arbre syntaxique en ajoutant un token à chaque étape, en s'appuyant sur un GNN pour encoder l'arbre partiel et prédire l'action suivante. Cette méthode a montré des performances compétitives sur des corpus tels que le Penn Treebank et le Chinese Treebank.

c) Des usages croissants en machine learning

Le principe du Machine Learning est simple : transformer un problème en la transformation d'une entrée en prédiction. Prendre son objet d'entrée, le transformer en vecteur (feature engineering) et entraîner le modèle à prédire la sortie attendue. On peut donc encoder des objets complexes (molécules, gènes…) et aussi des graphes d'où l'émergence d'une série de travaux consacrés au machine learning appliqué aux graphes[10]. Il faut distinguer le cas où le graphe est l'objet de la prédiction du système est un élément du graphe (ex pour le parsing syntaxique où on peut transformer la prédiction d'une arrête en un problème de classification binaire, et la catégorie de la dépendance en un problème de classification multiclasse) mais dont les embeddings sont faits à partir d'autre chose (texte de la phrase + mot + POS+ sa position + relations déjà prédites) des cas où c'est le graphe qui est encodé vectoriellement (et on ne prédit pas forcément des éléments liés au

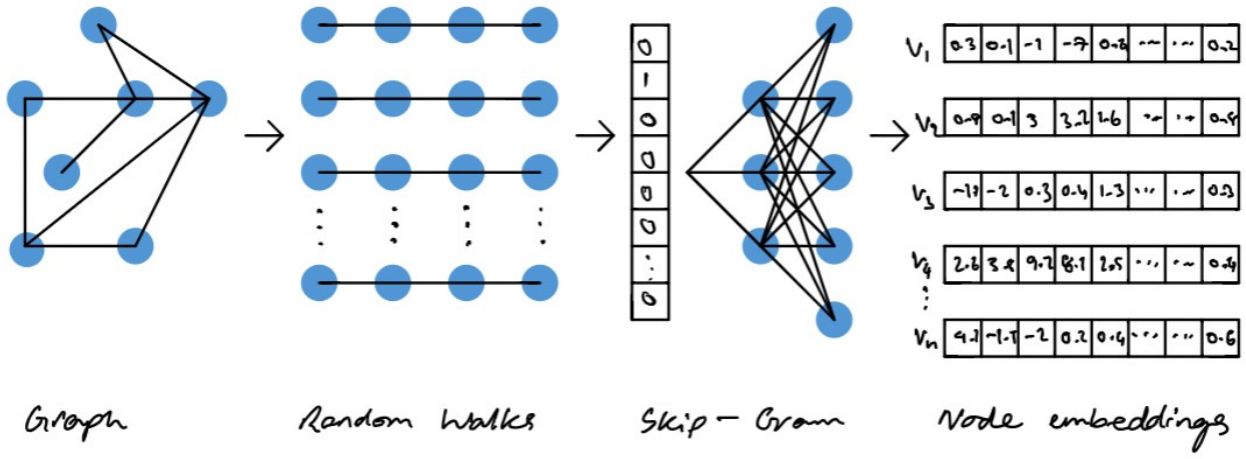
10 https://isamu-website.medium.com/understanding-graph-machine-learning-in-the-era-of-large-language-models-llms-dce2fd3f3af4 Pour une introduction pas trop violente et bien illustrée. Sinon https://www.manning.com/books/graph-powered-machine-learning. Je peux le prêter si besoin.

graphe, on peut prédire une décision binaire, comme « fraude » ou « pas fraude »). Dans le premier cas, on ne fait qu'encoder des objets, mais pas les relations qu'ils entretiennent avec les autres dans le graphe. Le but du Graph Machine Learning est précisément de capturer dans l'espace vectoriel latent les associations avec les objets, qui sont parfois plus importantes que les objets eux-mêmes. Le GML a connu plusieurs enjeux techniques qu'il a fallu dépasser.

* L'embedding des nœuds.

Dans un graphe, les nœuds représentent parfois des objets qui contiennent des informations que l'on peut transformer en vecteur. Par exemple pour construire un système de recommandation de films ou de série, on peut utiliser des informations sur le film (résumé, acteurs, réalisateur) pour créer un embedding de ces éléments textuels. On crée donc le système de recommandation à partir du contenu sémantique dont on dispose sur les films, et pas du comportement des utilisateurs des plateformes. Mais tous les films d'espions, ou tous les films d'Almodovar ne se valent pas. Il serait donc plus intéressant pour conseiller un film de regarder ce que les utilisateurs qui ont les mêmes goûts ont regardé, et ces informations sont stockées dans un graphe.

L'objectif principal est donc de capturer la topologie, la structure de ce graphe dans l'embedding des nœuds. Une intuition des chercheurs de Stanford en 2016 a été de s'inspirer de la démarche entreprise en NLP 3 ans plus tôt pour faire avancer les embeddings de mots. L'idée principale est que, de même qu'un mot est défini par ses voisins (Mikolov et al., 2013), dans un graphe, précisément un nœud est défini par ses voisins, par ses connections avec d'autre nœuds. Mais comment linéariser un graphe complexe ? L'intuition géniale qui a permis d'adapter Word2vec aux graphes est de dire qu'il n'est pas forcément nécessaire de donner le graphe en un bloc au système, mais qu'il était plus réalisable de le faire par petits bouts. Il est possible de décrire des sections d'un graphes en faisant ce qu'on appelle une marche aléatoire (random walk)[11] : on part d'un nœud au hasard, puis on va vers un de ses voisins, et ainsi de suite en s'arrêtant à un moment donné.



On construit donc des chemins e1, e2… en de n nœuds que l'on peut assimiler à des phrases de n mots, et ensuite utiliser sur ce « corpus » la stratégie d'embedding de word2vec pour obtenir des vecteurs. Même si les pseudo phrases ne décrivent pas tout le graphe de façon unie, si on donne assez de phrases, la structure va se reconstituer dans l'espace latent. L'algorithme s'appelle donc naturellement… node2vec[12][13].

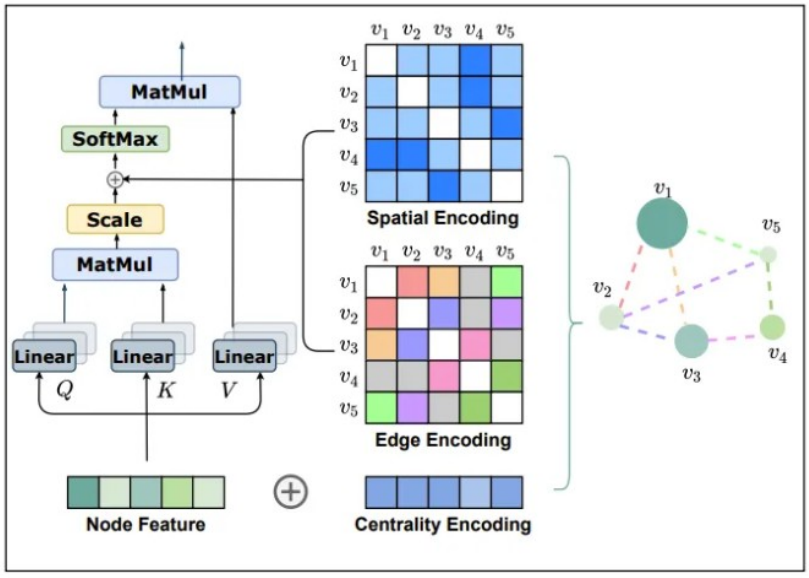
11 Pour une bonne explication visuelle https://towardsdatascience.com/complete-guide-to-understanding-node2vec-algorithm-4e9a35e5d147

12 https://arxiv.org/abs/1607.00653

13 https://medium.com/towards-data-science/node2vec-explained-db86a319e9ab pour l'illustration

Pour préciser : les marches aléatoires ne le sont pas entièrement. Elles sont dites biaisées et conditionnées en équilibrant deux paramètres :
Paramètre p : contrôle la probabilité de revenir au nœud précédent, influençant ainsi la profondeur de l'exploration locale.
Paramètre q : détermine la propension à explorer des zones plus éloignées du graphe, affectant l'étendue de l'exploration globale.
Le but de ces paramètres de contrôle est de capturer l'homophilie (similarité locale) : Les nœuds proches ou directement connectés sont explorés et la similarité structurelle : Les nœuds ayant des rôles similaires dans le graphe, même s'ils sont éloignés, sont également pris en compte. Les similarités locales et structurelles sont prises en compte, en opposition à des travaux précédents comme Deepwalk qui utilisaient des marches aléatoires classiques. Cela implique que p et q deviennent des hyperparamètres important de l'algorithme.

Boosté par cette belle réalisation, le Machine Learning sur les graphes a suivi les différentes révolutions des architectures et a tenté d'élargir ses cas d'application. En 2021, Microsoft research propose d'utiliser une architecture reposant sur l'atttention pour obtenir des représentations contextuelles des nœuds qui ne seraient plus dépendantes des marches aléatoires. L'article "Do Transformers Really Perform Bad for Graph Representation?[14]" présente **Graphormer**, une architecture Transformer adaptée à l'apprentissage de représentations de graphes. Cette approche intègre des encodages structurels spécifiques pour capturer efficacement les informations topologiques des graphes.



Encodage de Centralité : Chaque nœud se voit attribuer un vecteur appris basé sur sa centralité, mesurée par son degré. Cet encodage reflète l'importance relative des nœuds, permettant au modèle de distinguer les rôles structuraux au sein du graphe.

Encodage Spatial : Pour chaque paire de nœuds, un biais est introduit dans le mécanisme d'attention en fonction de la distance du plus court chemin les séparant. Cette méthode permet au modèle de capturer les relations structurelles entre nœuds, indépendamment de leur proximité immédiate.

Encodage des Arêtes : Les caractéristiques des arêtes sont incorporées en calculant une moyenne pondérée des produits scalaires entre les caractéristiques des arêtes et des vecteurs appris le long du plus court chemin entre deux nœuds. Cette technique enrichit la représentation en tenant compte des informations spécifiques aux connexions.

14 https://arxiv.org/abs/2106.05234

En combinant[15] ces encodages, Graphormer adapte le mécanisme d'attention des Transformers pour qu'il prenne en compte les structures complexes des graphes, améliorant ainsi les performances sur diverses tâches d'apprentissage de représentations de graphes

D’autres tentatives ont était faites avec des GANs, de l’apprentissage supervisé… Des approches à la BERT (next word prediction) étant bien sur impossible car on ne dispose pas d’un grand « corpus de graphes » et que chaque graphe n’a de sens que par rapport à son cas d’usage limité. C’est là le principal défi de la discipline ; alors que les corpus de texte sont largement disponibles et permettent d’entraîner un puissant modèle de fondation avant le fine tuning sur une tâche de spécialité, c’est plus difficile sur des graphes où des objets semblables peuvent selon le cas d’usage représenter des situations bien différentes. Des travaux intéressants avancent toutefois dans ce sens.[16] Ces embeddings de graphe ne capturent le plus souvent que la structure du graphe indépendamment du sémantisme des neouds et des relations. Cela fonctionne bien quand les nœuds et relations représentent la même chose, moins quand ils sont très hétérogène et donc quand le sémantisme du nœud importe beaucoup. L’autre défi réside à fusionner dans l’embedding les informations sur la structures du graphe et celles sur le sens des nœuds.

15 Source de l’illustration : https://isamu-website.medium.com/understanding-graph-machine-learning-in-the-era-of-large-language-models-llms-dce2fd3f3af4

16 https://arxiv.org/abs/2310.00149