# Application des algorithmes de fouille de graphe aux forums d'entraide

## 3.1 Introduction

Dans ce chapitre, nous présentons les algorithmes de fouille de graphes et leurs familles appliqués au forum d'entraide en ligne. Ces algorithmes sont la base de système d'analyse, nous permettant d'atteindre nos fins et d'extraire les connaissances cachés et invisibles. Dans ce qui suit, nous détaillons chaque algorithme et nous montrons comment l'appliquer au système d'analyse. Notons que les algorithmes que nous détaillons dans ce chapitre sont eux appartenant à la bibliothèque Graph Data Science de l'éditeur Neo4j.

## 3.2 Les algorithmes de fouille de graphe utilisés et leurs familles

Les algorithmes existent dans l'un des trois niveaux de maturité [43] :

- **Qualité de production :** Indique que l'algorithme a été testé en termes de stabilité et d'évolutivité. Les algorithmes de ce niveau sont préfixés par **gds.<algorithm>**.
- **Bêta :** Indique que l'algorithme est candidat au niveau de qualité de production. Les algorithmes de ce niveau sont préfixés par **gds.beta.<algorithm>**.
- **Alpha :** Indique que l'algorithme est expérimental et peut être modifié ou supprimé à tout moment. Les algorithmes de ce niveau sont préfixés par **gds.alpha.<algorithm>**.

### 3.2.1 Détection de communauté

Une communauté est un groupe de personnes, d'objets ou de concepts qui entretiennent des liens privilégiés parce qu'ils ont des affinités particulières, ou présentent des caractéristiques similaires, ou encore partagent des centres d'intérêts, etc. Au sens du graphe, une communauté est un cluster de nœuds qui sont fortement liés entre eux, et faiblement liés avec les nœuds situés en dehors de la communauté.

La bibliothèque Neo4j Graph Data Science (GDS) comprend plusieurs algorithmes de détection de communauté, mais selon l'utilité de chacun et notre objectif ainsi que le niveau de maturité, nous avons utilisé les algorithmes suivants :

#### 3.2.1.1 Louvain

C'est un algorithme qui appartient au niveau -Qualité de production-, sa méthode est basée sur la maximisation de modularité , où la modularité [1] quantifie la qualité d'une affectation de nœuds aux communautés. Louvain se caractérise par limite de résolution et une scalabilité bien meilleure. L'algorithme de Louvain est un algorithme de clustering hiérarchique, qui fusionne récursivement les communautés en un seul nœud et exécute le clustering de modularité sur les graphes condensés, mais peut également s'exécuter sur des graphes pondérés, prenant en compte les poids de relation donnés lors du calcul de la modularité . [45]

**A- L'algorithme**

**Etape 1 :**
**1-** chaque nœud du graphe est affecté à sa propre communauté.
**2-** pour chaque nœud « i », on calcule le changement de modularité occasionné par la suppression de « i » de sa propre communauté et son déplacement dans la communauté de chacun des voisins « j » de « i » ( « i » est placé dans la communauté qui entraine la plus grande augmentation de la modularité. Si aucune augmentation n'est possible, « i » reste dans sa communauté d'origine).
Ce processus est appliqué de manière répétée et séquentielle sur tous les nœuds jusqu'à ce qu'aucune augmentation de la modularité ne se produise.
**Etape 2 :**
Tous les nœuds d'une même communauté sont regroupés, et un nouveau graphe est construit où les nœuds sont les communautés de la phase précédente. [66]

**B- Syntaxe d'exécution**

**Call gds.louvain.stream (configuration : Map)**
**YIELD**
**nodeId : Integer,**
**communityId : Integer,**
**intermediateCommunityIds : Integer [] [45]**

**C- Configuration**

Dans le tableau 3.1, nous présentons une configuration générale pour l'exécution de l'algorithme :

1. Elle mesure la différence entre le nombre d'arêtes internes à la communauté et l'espérance de cette valeur dans le modèle de configuration

| Nom | Type | Valeur par défaut | Optionnel | Description |
|---|---|---|---|---|
| nodeProjection | String, String [] ou Map | Null | Oui | Projection de nœud utilisée pour la création de graphes anonymes via une projection native. |
| relationProjection | String, String [] ou Map | Null | Oui | Projection de relation utilisée pour la création de graphe anonyme une projection native. |
| nodeProperties | String, String [] ou Map | Null | Oui | Propriétés du nœud à projeter lors de la création d'un graphe anonyme. |
| relationProperties | String, String [] ou Map | Null | Oui | Propriétés de la relation à projeter lors de la création d'un graphe anonyme. |

TABLE 3.1: Configuration de l'algorithme louvain [45]

#### 3.2.1.2 Composants fortement connectés

Un ensemble est considéré comme un composant fortement connecté s'il existe un chemin dirigé entre chaque couple de nœuds au sein de l'ensemble. L'algorithme permet de calculer la décomposition d'un graphe à un ensemble des composantes fortement connexes. Cet algorithme appartient au niveau -Alpha-. [48]

**A- L'algorithme**

Une application classique de l'algorithme de recherche en profondeur d'abord, un parcours de graphe qui commence à un nœud donné et explore autant que possible le long de chaque branche avant de revenir en arrière.
**1-** Marquer v comme sommet déjà visité.
**2- Pour toutes** les arêtes dirigées de v à un sommet w, **si** le sommet w n'est pas marqué comme déjà visité, **alors** ajouter w au composant de v et aller à 1 pour w.
**3-** commençant une nouvelle recherche en profondeur chaque fois que la boucle atteint un sommet qui n'a pas déjà été ajouté à un composant trouvé trouvé précédemment. [65]

**B- Syntaxe d'exécutionL'algorithme**

**CALL gds.alpha.scc.stream (configuration : Map)**
**YIELD nodeId, compoinent** [48]

**C- Configuration**

Dans le tableau 3.2, nous présentons une configuration générale pour l'exécution de l'algorithme :

| Nom | Type | Valeur par défaut | Optionnel | Description |
|---|---|---|---|---|
| nodeProjection | String, String [] ou Map | Null | Oui | Projection de nœud utilisée pour la création de graphes anonymes via une projection native. |
| relationProjection | String, String [] ou Map | Null | Oui | Projection de relation utilisée pour la création de graphe anonyme une projection native. |
| nodeProperties | String, String [] ou Map | Null | Oui | Propriétés du nœud à projeter lors de la création d'un graphe anonyme. |
| relationProperties | String, String [] ou Map | Null | Oui | Propriétés de la relation à projeter lors de la création d'un graphe anonyme. |

TABLE 3.2: Configuration de l'algorithme Composants fortement connectés [48]

#### 3.2.1.3 Propagation d'étiquettes

L'algorithme de propagation d'étiquettes (Label Propagation Algorithm - LPA) est un algorithme rapide, appartenant au niveau -Qualité de production-, et détecte ces communautés en utilisant uniquement la structure du graphe comme guide, et ne nécessite pas de fonction objective prédéfinie ou d'informations préalables sur les communautés. [44]

**A- L'algorithme**

**1-** Chaque nœud est initialisé avec une étiquette de communauté unique (un identifiant).
**2-** Ces étiquettes se propagent à travers le réseau.
**3-** À chaque itération de propagation, chaque nœud met à jour son étiquette à celle à laquelle appartient le nombre maximum de ses voisins. Les liens sont rompus de manière arbitraire mais déterministe.
**4-** LPA atteint la convergence lorsque chaque nœud a l'étiquette de la majorité de ses voisins.
**5-** LPA s'arrête si la convergence ou le nombre maximal d'itérations défini par l'utilisateur est atteint. [44]

**B- Syntaxe d'exécution**

**CALL gds.labelPropagation.stream(configuration : Map)**
**YIELD**
**nodeId : Integer, communityId : Integer** [44]

**C- Configuration**

Dans le tableau 3.3, nous présentons une configuration générale pour l'exécution de l'algorithme :

| Nom | Type | Valeur par défaut | Optionnel | Description |
|---|---|---|---|---|
| nodeProjection | String, String [] ou Map | Null | Oui | Projection de nœud utilisée pour la création de graphes anonymes via une projection native. |
| relationProjection | String, String [] ou Map | Null | Oui | Projection de relation utilisée pour la création de graphe anonyme une projection native. |
| nodeProperties | String, String [] ou Map | Null | Oui | Propriétés du nœud à projeter lors de la création d'un graphe anonyme. |
| relationProperties | String, String [] ou Map | Null | Oui | Propriétés de la relation à projeter lors de la création d'un graphe anonyme. |
| maxIterations | Entier | Dix | Oui | Nombre maximal d'itérations à exécuter. |
| nodeWeightProperty | Chaîne | Null | Oui | Nom de propriété du nœud contenant le poids. Doit être numérique. |
| relationWeightProperty | String | Null | Oui | Le nom de propriété qui contient le poids. |
| seedProperty | String | n/a | Oui | Utilisé pour définir le jeu initial d'étiquettes (doit être un nombre). |

TABLE 3.3: Configuration de l'algorithme propagation d'étiquettes [44]

### 3.2.2 Centralité

Les algorithmes de centralité sont utilisés pour déterminer l'importance de nœuds distincts dans un réseau. Nous avons utilisé les algorithmes de centralité suivants :

#### 3.2.2.1 Page Rank (Classement)

Le Page Rank est un algorithme qui appartient au niveau -Qualité de production- et qui mesure l'influence ou l'importance des nœuds dans un graphe orienté, il est calculé en fonction du nombre de relations entrantes et de l'importance des nœuds sources correspondants.
Le Page Rank est défini dans le document Google d'origine comme une fonction qui résout l'équation suivante modularité [46] :

$$\mathbf{PR(A) = (1\text{-}d) + d\ (PR(T1)/C(T1) + \ldots.. + PR(Tn)/C(Tn))}$$

Où,

- **A :** une page qui possède des pages (T1, T2,...,Tn) auxquelles elle pointe (c.-à-d. sont des citations).
- **d :** un facteur d'amortissement qui peut être réglé entre 0 et 1, il est généralement réglé sur 0,85.
- **C(A) :** est défini comme le nombre de liens sortant de la page A.

### A- L'algorithme

**1.** initialisez toutes les composantes de **V** à **1/N** (**N** étant le nombre des nœuds du graphe et **V** le vecteur de répartition sur le graphe )
**2.** calculez **V'= V*M**, (où **M** est la matrice représentant le graphe)
**3.** copiez le contenu de **V'** dans **V**.
**4.** reprenez en 2 sauf si les deux vecteurs **V** et **V'** sont très proches, dans ce cas l'algorithme est terminé. [3]

### B- Syntaxe d'exécution

**Call gds.pageRank.stream (configuration : Map)**
**YIELD**
**nodeId : Integer,**
**score : float** [46]

### C- Configuration

Dans le tableau 3.4, nous présentons une configuration générale pour l'exécution de l'algorithme :

| Nom | Type | Valeur par défaut | Optionnel | Description |
|---|---|---|---|---|
| nodeProjection | String, String [] ou Map | Null | Oui | Projection de nœud utilisée pour la création de graphes anonymes via une projection native. |
| relationProjection | String, String [] ou Map | Null | Oui | Projection de relation utilisée pour la création de graphe anonyme via une projection native. |
| nodeProperties | String, String [] ou Map | Null | Oui | Propriétés du nœud à projeter lors de la création d'un graphe anonyme. |
| relationProperties | String, String [] ou Map | Null | Oui | Propriétés de la relation à projeter lors de la création d'un graphe anonyme. |
| Facteur d'amortissement | Flotte | 0.85 | Oui | Le facteur d'amortissement du calcul du Page Rank |
| maxIterations | Entier | 20 | Oui | Nombre maximal d'itérations à exécuter. |
| relationWeightProperty | String | Null | Oui | Le nom de propriété qui contient le poids. |

TABLE 3.4: Configuration de l'algorithme Page Rank [46]

#### 3.2.2.2 Centralité de degré (degree)

L'algorithme de degré de centralité appartient au niveau -Beta-, il peut nous aider à trouver les nœuds les plus importantes et populaires dans un graphe. Il mesure le nombre de relations entrantes et sortantes d'un nœud (c'est-à-dire le nombre de voisins que possède un nœud).
Dans le cas d'un graphe orienté, on définit habituellement deux mesures distinctes de la centralité de degré, **le degré entrant** est souvent interprété comme une forme de popularité, et **le degré sortant** comme mesure du caractère grégaire (mesure d'influence).

La centralité degré du sommet **v** est défini comme [41] :

$$C_d(v) = \sum a(v,t)$$

Où : **a(v, t) = 1** si et seulement si le sommet **v** est lié au sommet **t**, **a(v,t)=0** autrement.

**A- Syntaxe d'exécution**

**CALL gds.alpha.degree.stream (configuration : Map)**
**YIELD node, score** [41]

**B- Configuration**

Dans le tableau 3.5, nous présentons une configuration générale pour l'exécution de l'algorithme :

| Nom | Type | Valeur par défaut | Optionnel | Description |
|---|---|---|---|---|
| nodeProjection | String, String [] ou Map | Null | Oui | Projection de nœud utilisée pour la création de graphes anonymes via une projection native. |
| relationProjection | String, String [] ou Map | Null | Oui | Projection de relation utilisée pour la création de graphe anonyme une projection native. |
| WeightProperty | String | Null | Oui | Le nom de propriété qui contient le poids. S'il est nul, traite le graphe comme non pondéré. Doit être numérique |
| Default value | float | 0.0 | Oui | La valeur par défaut du poids au cas où il serait manquant ou invalide |

TABLE 3.5: Configuration de l'algorithme degré [41]

#### 3.2.2.3 Centralité d'interdépendance (Betweenness)

La centralité de l'interdépendance est un moyen de détecter les influenceurs sur le flux d'informations dans un graphe. Cet algorithme est souvent utilisé pour trouver des nœuds

qui servent de pont d'une partie d'un graphe à une autre. Il appartient au niveau -Alpha-Entre centralité peut être représenté par :

$$CB = \sum(NTC_{st}(v)/NTC_{st})$$

Où,

- $NTC$ **:** le nombre total des plus courts chemins du nœud « s » au nœud « t »
- $NTC_{st}(v)$ **:** le nombre de tels chemins qui passent par « v ».

On peut normaliser l'intermédiarité en divisant le résultat par le nombre de couples ne comprenant pas v, nombre qui est égal à **(N-1)(N-2)** dans un graphe orienté à N sommets et à **(N-1)(N-2)/ 2** dans un graphe non orienté. [39]

**A- L'algorithme**

**1.** Pour chaque couple (s,t), on calcule les plus courts chemins les reliant.
**2.** Pour chaque couple (s,t), on détermine la proportion de plus cours chemins qui passent par le sommet v.
**3.** On somme cette fraction sur tous les couples (s,t) de sommet. [64]

**B- Syntaxe d'exécution**

**CALL gds.alpha.betweenness.stream (configuration : Map)**
**YIELD nodeId, Centrality.** [39]

**C- Configuration**

Dans le tableau 3.6, nous présentons une configuration générale pour l'exécution de l'algorithme :

| Nom | Type | Valeur par défaut | Optionnel | Description |
|---|---|---|---|---|
| nodeProjection | String, String [] ou Map | Null | Oui | Projection de nœud utilisée pour la création de graphes anonymes via une projection native. |
| relationProjection | String, String [] ou Map | Null | Oui | Projection de relation utilisée pour la création de graphe anonyme une projection native. |
| Stratégie | String | 'Aléatoire' | Oui | La stratégie de sélection des nœuds. |
| Probabilité | flotte | log10 (N) / e ^2 | Oui | La probabilité qu'un nœud soit sélectionné. Valeurs comprises entre 0 et 1. |

TABLE 3.6: Configuration de l'algorithme Betweenness [39]

#### 3.2.2.4 Eingenvector centrality

La centralité d'Eigenvector est un algorithme qui appartient au niveau -Alpha- et qui mesure l'influence transitive ou la connectivité des nœuds. Il considère l'importance transitive d'un nœud dans un graphe, plutôt que de considérer uniquement son importance directe. [42]. C'est-à-dire il mesure l'importance d'un nœud tout en tenant compte de l'importance de ses voisins.
Le score de centralité relative (centralité d'Eigenvector) du sommet v est défini comme :

$$X_v = 1/\lambda \sum_{t \in M(v)} (xt = 1/\lambda \sum_{t \in G} (a_{v,t} x_t))$$

Où,

- **G :** c'et le graphe.
- **A($a_{v,t}$) :** la matrice représentant le graphe , $a_{v,t} = \mathbf{1}$ si le sommet **v** est lié au sommet **t** et $a_{v,t} = \mathbf{0}$ autrement.
- **M(v) :** c'et l'ensemble des voisins de **v** .
- $\lambda$ **:** est une constante. [21]

**A- L'algorithme**

La documentation de Neo4j ne fournit pas de détails sur l'algorithme.

**B- Syntaxe d'exécution**

**CALL gds.alpha.eigenvector.stream (configuration : Map)**
**YIELD node, score** [42]

**C- Configuration**

Dans le tableau 3.7, nous présentons une configuration générale pour l'exécution de l'algorithme :

2. Normaliser les scores pour qu'ils totalisent 1
3. Divisez chaque score par la racine carrée de la somme au carré de tous les scores

| Nom | Type | Valeur par défaut | Optionnel | Description |
|---|---|---|---|---|
| nodeProjection | String, String [] ou Map | Null | Oui | Projection de nœud utilisée pour la création de graphes anonymes via une projection native. |
| relationProjection | String, String [] ou Map | Null | Oui | Projection de relation utilisée pour la création de graphe anonyme une projection native. |
| relationWeightProperty | String | Null | Oui | Nom de la propriété de relation qui contient le poids. Si null, traite le graphique comme non pondéré. Doit être numérique. |
| Normalisation | String | Null | Oui | Le type de normalisation à appliquer aux résultats. Les valeurs valides sont max, 11norm [2] , 12norm [3] |
| maxIterations | Int | 20 | Oui | Nombre maximal d'itérations de EigenvectorCentrality à exécuter. |
| sourceNodes | liste <noeud> | liste vide | Oui | Une liste de f nœuds pour commencer le calcul. |

TABLE 3.7: Configuration de l'algorithme Eingenvector centrality [42]

### 3.2.3 Similarité

La similarité est un des critères pour l'analyse informatique de clusters et pour le partitionnement de données, Les algorithmes de similarité calculent la similitude de couples de nœuds à l'aide de différentes métriques vectorielles.

L'algorithme de similarité des nœuds appartient au niveau -Qualité de production-, il compare un ensemble de nœuds en fonction des nœuds auxquels ils sont connectés , par couple sur la base de la métrique Jaccard :

$J(a,b) = (A \cap B)/(A \cup B)$
Où : **A, B :** sont des ensembles de noeuds connctés aux noeuds a et b. [47]

**A- L'algorithme**

**1.** calculer les similarités entre les nœuds .
**2.** trier les couples des nœuds dans l'ordre décroissant.
**3.** donner comme réponse la liste des N premiers nœuds, où N est par exemple fixé par l'utilisateur. [71]

**B- Syntaxe d'exécution**

**CALL gds.nodeSimilarity.stream (configuration : Map)**
**YIELD**
**node1 : Integer, node2 : Integer, similarity : Float** [47]

**C- Configuration**

Dans le tableau 3.8, nous présentons une configuration générale pour l'exécution de l'algorithme :

| Nom | Type | Valeur par défaut | Optionnel | Description |
|---|---|---|---|---|
| nodeProjection | String, String [] ou Map | Null | Oui | Projection de nœud utilisée pour la création de graphes anonymes via une projection native. |
| relationProjection | String, String [] ou Map | Null | Oui | Projection de relation utilisée pour la création de graphe anonyme une projection native. |
| nodeProperties | String, String [] ou Map | Null | Oui | Propriétés du nœud à projeter lors de la création d'un graphe anonyme. |
| relationProperties | String, String [] ou Map | Null | Oui | Propriétés de la relation à projeter lors de la création d'un graphe anonyme. |
| writeProperty | String | n/a | Non | Mode WRITE uniquement : propriété de relation dans laquelle le score de similitude est écrit. |

TABLE 3.8: Configuration de l'algorithme Similarité des noeuds [47]

### 3.2.4 Prédictions des liens

La prédiction de liens peu être utilisée pour la reconstitution de liens manquants sur un graphe, c'est de construire un réseau d'interactions basées sur des données observables et essayer ensuite de déduire des liens supplémentaires qui ne sont pas directement visibles dans le graphe en main mais qui peuvent exister dans le futur.

#### 3.2.4.1 Adamic Adar

L'algorithme Adamic Adar appartient au niveau -Alpha-,il est utilisé pour prédire les liens dans un graphe, en fonction de la quantité de liens partagés entre deux nœuds, à l'aide de la formule suivante :

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} (1/Log(N(u)))$$

Où : **N(u) :** l'ensemble des nœuds adjacents à « u ». [38]

**A- L'algorithme**

La documentation de Neo4j ne fournit pas de détails sur l'algorithme.

**B- Syntaxe d'exécution**

**RETURN gds.alpha.linkprediction.adamicAdar(node1 :Node, node2 :Node, relationshipQuery :String, direction :String )** [38]

**C- Configuration**

Dans le tableau 3.9, nous présentons une configuration générale pour l'exécution de l'algorithme :

| Nom | Type | Valeur par défaut | Optionnel | Description |
|---|---|---|---|---|
| node1 | Nœud | Null | Non | Un nœud |
| node2 | Nœud | Null | Non | Un autre noeud |
| relationshipQuery | String | Null | Oui | Le type de relation utilisé pour calculer la similitude entre node1 et node2 |
| Direction | String | tous les deux | Oui | La direction du type de relation utilisée pour calculer la similitude entre node1 et node2 |

TABLE 3.9: Configuration de l'algorithme Adamic Adar [38]

#### 3.2.4.2 Voisins communs

L'algorithme des voisins communs est basé sur l'idée que deux nœuds qui sont connectés à un nœud en commun sont plus susceptibles d'être en interaction plus que ceux qui n'ont pas des nœuds en commun. Cet algorithme appartient au niveau -Alpha-. Il est calculé à l'aide de la formule suivante :

$$NC(x, y) =\mid N(x) \cap N(y) \mid$$

Où : **N(x), N (y)** : est l'ensemble des nœuds adjacents au nœuds x et y. [40]

**A- L'algorithme**

La documentation de Neo4j ne fournit pas de détails sur l'algorithme.

**B- Syntaxe d'exécution**

**RETURN gds.alpha.linkprediction.commonNeighbors (node1 :Node, node2 :Node, relationshipQuery : String, direction :String)** [40]

**C- Configuration**

Dans le tableau 3.10, nous présentons une configuration générale pour l'exécution de l'algorithme :

| Nom | Type | Valeur par défaut | Optionnel | Description |
|---|---|---|---|---|
| node1 | Nœud | Null | Non | Un nœud |
| node2 | Nœud | Null | Non | Un autre noeud |
| relationshipQuery | String | Null | Oui | Le type de relation utilisé pour calculer la similitude entre node1 et node2 |
| Direction | String | tous les deux | Oui | La direction du type de relation utilisée pour calculer la similitude entre node1 et node2 |

TABLE 3.10: Configuration de l'algorithme Voisins communs [40]

## 3.3 Modélisation orientée graphes de forums d'entraides

Pour atteindre l'objectif d'application des algorithmes de fouille de graphe aux forum de Q/R, nous devons d'abord représenter les données du forum par un graphe et établir les correspondances entre les données d"origine et les données du graphe au niveau schéma. Nous commençons d'abord par présenter le modèle de graphes puis les mises en correspondances.

### 3.3.1 Les nœuds

Nous avons extrait deux types de nœuds **(labels)** :

| Nœud | Désignation | Propriété | Type de propriété | Désignation de propriété |
|---|---|---|---|---|
| **Users** | Les tilisateurs de forum | **userId** | Integer | identifient de l'utilisateur |
| | | **userName** | String | le nom de l'utilisateur |
| **Tags** | Les tags | **tagId** | Integer | identifient de les tags |
| | | **tagName** | String | le nom de tag |

TABLE 3.11: Table des nœuds

### 3.3.2 Les relations

Nous nous sommes interessées à 7 types de relations entre les noeuds :

| Relation | Désignation | Propriété | Type de propriété | Désignation de propriété |
|---|---|---|---|---|
| **Comment** | défini le fait qu'un utilisateur commente une publication d'un utilisateur | **NbrOfComments** | Integer | indique le nombre des commentaires |
| **UpVote, DownVote et FavoriteVote** | indique trois type de vote, bon vote et mauvais vote, vote préférée, fait d'un utilisateur sur un utilisateur | **NbrOfUpvotes** | Integer | le nombre des votes positifs fait par un utilisateur |
| | | **NbrOfDownVotes** | Integer | le nombre des votes négatifs fait par un utilisateur |
| | | **NbrOfFavoriteVotes** | Integer | le nombre des votes préférés fait par un utilisateur. |
| **Answer** | c'est la réponse d'un utilisateur | **NbrOfAnswers** | Integer | désigne le nombre des réponses proposés par un utilisateur pour les questions d'un utilisateur |
| **LieeA** | le lien entre les tags | **NbrOfOccurences** | Integer | indique le nombre des fois qu'un tag est mentionné avec un autre. |
| **RelatedLink** | le lien entre les utilisateurs | **NbrOfLinks** | Integer | indique le nombre des liens entre les utilisateurs. |

TABLE 3.12: Table des relations

### 3.3.3 Le schema

La figure 3.1, illustre le schéma orienté graphe des données des Q/R.

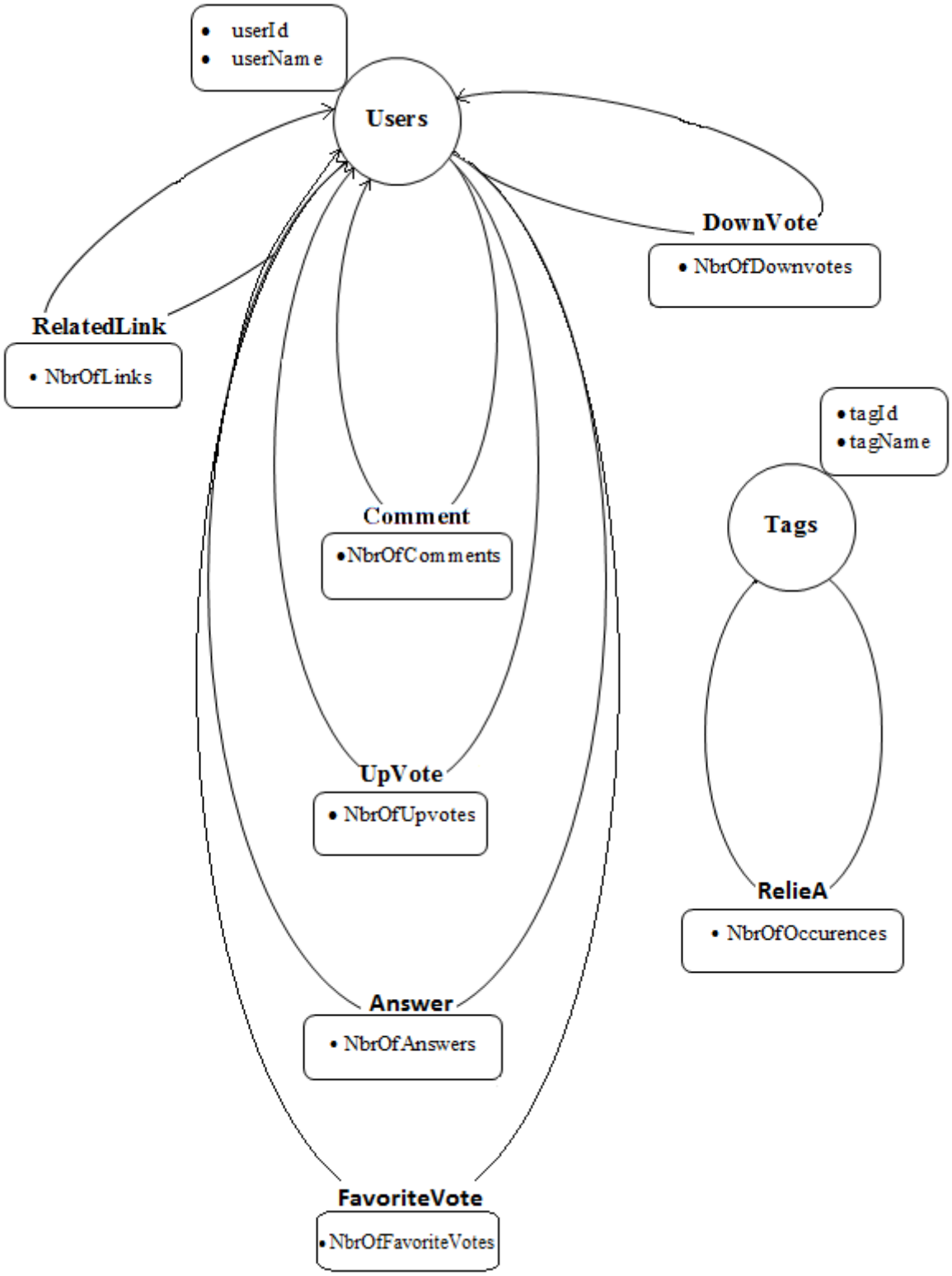


FIGURE 3.1: Le schéma de la base de données orientée graphe

## 3.4 Correspondances entre le schéma relationnel et le schéma Graphe

Dans le chapitre précédent, nous avons présenté le schéma des données de notre sources (la base relationnelle) ainsi que le schéma orienté graphe dans la section précédente. Dans le tableau 3.13 nous présentons la mise en correspondance entre les deux schémas de bases de données (mapping), les opérations nécessaires entre les deux schémas. Le tableau est divisé en deux grande colonnes schéma orienté graphe et schéma relationel. La colonne schéma orienté graphe est subdivisée en trois colonnes nom de composant, type de composant, propriété. Les sous-colonnes de la colonne schéma relationnel sont subdivisées en deux souscolonnes : nom de la table, et nom de la colonne. Dans la colonne description, nous indiquons si la mise en correspondance est direct ou nécessite une opération de transformation.

| **Schéma orienté graphe** | | | **Schéma relationnel** | | **Description (mise en correspondance)** |
|---|---|---|---|---|---|
| **Nom de composant** | **type** | **Propriété** | **Nom de table** | **Nom de la colonne** | |
| Users | Nœud (label) | userId | Users | Id | Mappage direct |
| | | userName | | DisplayName | Mappage direct |
| Tags | Nœud | tagId | Tags | TagId | Mappage direct |
| | | tagName | | TagName | Mappage direct |
| Comment | Relation (Relationship) | NbrOfComments | Comments, Posts | UserId, OwnerUserId | Jointure de la table comments avec la table posts et groupement (Group By) |
| Answer | Relation | NbrOfAnswers | Posts | OwnerUserId | Jointure de la table posts avec elle-même et groupement (Group By) |
| UpVote | Relation | NbrOfUpvotes | Votes, Posts | UserId, OwnerUserId | Jointure de la table votes avec la table posts et groupement (Group By) |
| DownVote | Relation | NbrOfDownVotes | Votes, Posts | UserId, OwnerUserId | Jointure de la table votes avec la table posts et groupement (Group By) |
| FavoriteVote | Relation | NbrOfFavoriteVotes | Votes, Posts | UserId, OwnerUserId | Jointure de la table votes avec la table posts et groupement (Group By) |
| LiéA | Relation | NbrOfOccurences | Post, Tags | TagId | Jointure de la table posttags avec elle-même et groupement (Group By) |
| RelatedLink | Relation | NbrOfLinks | Posts | UserId, OwnerUserId | Double Jointure de la table posts avec la table postlinks et groupement (Group By) |

Table 3.13: Correspondances entre le schéma relationnel et le schéma orienté graphe

## 3.5 Plan d'application des algorithmes de fouille de Graphes

L'application des algorithmes de graphe de fouille de données se fait par un appel d'exécution directe du code de lancement de l'algorithme avec une configuration et des paramètres données, comme nous avons vu précédemment.
Dans cette section, nous présentons un plan d'application des algorithmes dans le tableau suivant (table 3.14). Cette table est divisée en quatre grandes lignes représentant les familles des algorithmes de graphe Centralité, Communauté, Similarité, et Prédiction des liens. Pour chaque famille nous présentons l'algorithme utilisé, l'objectif concrêt de l'utilisation de l'algorithme, le nœud concerné par l'analyse, la relation, une colonne pour indiquer si la relation et orienté ou non, et une autre pour le poids.

| Famille d'algorithme | Algorithme | Objectif Concret | Nœuds | Relations | Relation orientée ? | Poids |
|---|---|---|---|---|---|---|
| Centralité | Degré | Connaitre les users qui ont plus d'interaction avec les autres | Users | Toutes | Oui | / |
| | | Connaitre les users qui ont obtenu le plus de Upvotes | Users | UpVote | Oui | NbrOfUpvotes |
| | | Connaitre les users qui ont obtenu les plus de Downvotes | Users | DownVote | Oui | NbrOfDownvotes |
| | | Connaitre les users qui ont obtenu le plus de Favoritevotes | Users | FavoriteVote | Oui | NbrOfFavoriteVotes |
| | | Connaitre les users qui répondent ou à qui on répond le plus | Users | Answer | Oui | NbrOfAnswers |
| | | Connaitre les users qui commentent ou qu'on comment le plus | Users | Comment | Oui | NbrOfComments |
| | Page Rank | Connaitre les users qui ont plus d'influence de Upvote | Users | UpVote | Oui | NbrOfUpvotes |
| | | Connaitre les users qui ont plus d'influence de Downvote | Users | DownVote | Oui | NbrOfDownvotes |
| | | Connaitre les users qui ont plus d'influence de Favoritevote | Users | FavoriteVote | Oui | NbrOfFavoriteVotes |
| | | Connaitre les users qui ont plus d'influence de commentaires | Users | Comment | Oui | NbrOfComments |
| | | Connaitre les users qui ont plus d'influence de réponses | Users | Answer | Oui | NbrOfAnswers |
| | Betweennes | Identifier les utilisateurs importants pour la diffusion d'iformation (intermédiaire) avec lesquels ils devraient interagir à l'avenir | Users | Comment | Non | / |
| | Eingenvector centrality | Connaitre le mot clé le plus important (lié aux autres) | Tags | LieeA | Non | NbrOfOccurences |
| Détection des communautés | Louvain | Connaitre les groupes des users toutes interactions | Users | Toutes | Non | / |
| | | Connaitre les groupes de tags qui sont utilisée ensemble | Tags | LieeA | Non | NbrOfOccurences |
| | Composants fortement connectés | Connaitre les groupes des utilisateurs qui sont fortement liée | Users | RelatedLink | Non | NbrOfLinks |
| | propagation d'étiquettes | Connaitre les groupes des users qui commenter entre eux | Users | Comment | Non | NbrOfComments |
| Similarité | Similarité des nœuds | Connaitre les users qui ont des voisins communs (meme interaction) | Users | Comment | Oui | / |
| | | | Users | Answer | Oui | / |
| Prédiction des liens | Adamic Adar | Calculer la proximité de deux users pour prédire les liens | Users | RelatedLink | Non | / |
| | | | Users | Comment | Non | / |
| | | | Users | Answer | Non | / |
| | | | Users | Toutes | Non | / |
| | Voisins communs | Calculer la proximité entre deux nœuds | Users | Toutes | Non | / |

TABLE 3.14: Liste des algorithmes appliqués

## 3.6 Conclusion

Dans ce chapitre, nous avons abordé les algorithmes de graphe aux forums d'entraide utilisé dans notre travail, leurs familles, algorithmes, syntaxes et configurations. Aussi, nous avons présenté la modélisation orientée graphes de forums d'entraides en identifiants l'ensemble des nœuds et des relations à utiliser suivie par la présentation du schéma orientée graphe et la correspondance avec le schéma relationnel de la base du "StackExchange". Nous avons également défini le plan d'application des algorithmes de fouille de graphes avec leurs objectifs, les nœuds concernés, les relations et le poids.
En plus des algorithmes ci-dessus, il existe un grand nombre d'implémentations d'algorithmes développées dans le cadre de Neo4j Labs. Mais nous nous sommes contentées des algorithmes que nous avons jugés pertinents pour réaliser nos objectifs.