Découverte de connaissances à partir de grands graphes biologiques

Sabeur Aridhi

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Découverte de connaissances à partir de grands graphes biologiques

MÉMOIRE

présentée et soutenue publiquement le 18 décembre 2023

pour l'obtention d'une

Habilitation de l'Université de Lorraine

(mention informatique)

par

Sabeur Aridhi

Composition du jury

Rapporteurs :	Fatiha Sais, Professeur, LRI, Université Paris Saclay, France Philippe Fournier-Viger, Professeur, Shenzhen University, Chine Osmar Zaiane, Professeur, University of Alberta, Canada
Examinateurs :	Sarah Cohen Boulakia, Professeur, LRI, Université Paris-Saclay, France Abdoulay Baniré Diallo, Professeur, Université du Québec à Montréal, Canada Yannick Toussaint, Professeur, Loria, Université de Lorraine, France
Marraine scientifique :	Marie-Dominique Devignes, Chargée de recherche (HDR), CNRS, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Abstract

Graphs are used in many different fields, ranging from social networks computer security, geography and bioinformatics. Their versatility and their ability to model complex data make them a valuable tool for solving several problems especially in the domain of knowledge extraction from biological graphs which is our domain of interest. However, the use of graphs can present several challenges such as data representation, scalability and the dynamic/temporal behavior of some real world graphs. In the particular case of biological graphs, challenges related to the complexity of biological systems, interpretability and validation could be raised. Without achieving all of these challenges, we present in this manuscript several contributions. First, we present a novel protein-protein network (PPN) based approach which combines the notion of domain similarity with a graph neighborhood inference technique for protein function annotation. Then, we describe an explainable link prediction approach in biological knowledge graphs for drug repositioning. Finally, we present our contributions on distributed graph clustering and large-scale knowledge graph embedding.

Keywords: Protein function annotation, knowledge graphs, link prediction, structural clustering, knowledge graph embedding.

Résumé

Les graphes sont utilisés dans de nombreux domaines différents, allant de la sécurité informatique des réseaux sociaux, à la géographie et à la bioinformatique. Leur polyvalence et leur capacité à modéliser des données complexes en font un outil précieux pour résoudre plusieurs problèmes notamment dans le domaine de l'extraction de connaissances à partir de graphes biologiques qui est notre domaine d'intérêt. Cependant, l'utilisation de graphes peut présenter plusieurs défis tels que la représentation des données, l'évolutivité et le comportement dynamique/temporel de certains graphes réels. Dans le cas particulier des graphes biologiques, des défis liés à la complexité des systèmes biologiques, à l'interprétabilité et à la validation pourraient être soulevés. Sans s'attaquer à l'ensemble de ces défis majeurs, nous présentons dans ce manuscrit d'HDR plusieurs contributions. Tout d'abord, nous présentons une nouvelle approche basée sur un graphe de protéines qui combine la notion de similarité de domaine avec une technique d'inférence de voisinage de graphe pour l'annotation de la fonction des protéines. Ensuite, nous décrivons une approche de prédiction de liens explicables dans les graphes de connaissances biologiques pour le repositionnement des médicaments. Enfin, nous présentons nos contributions sur le clustering de graphes distribués et le plongement de graphes de connaissances à grande échelle.

Mots-clés: Annotation fonctionnelle de protéines, graphes de connaissances, prédiction de liens explicables, clustering structurel, plongement de graphes de connaissances.

Remerciements

Je tiens à remercier tout d'abord Fatiha Sais, Philippe Fournier-Viger et Osmar Zaiane pour avoir accepté de relire ce mémoire et pour l'avoir fait avec beaucoup de bienveillance. Merci également à Sarah Cohen-Boulakia, Abdoulay Baniré-Diallo et Yannick Toussaint pour avoir accepté d'examiner ce travail. Un très grand merci à Marie-Dominique Devignes pour le parrainage et pour tout l'accompagnement et les conseils précieux durant ces dernières années au sein de l'équipe Capsid. Le travail décrit dans ce mémoire d'HDR n'est pas le fruit de mon seul travail, mais celui de nombreux échanges et collaborations. Je remercie Manel, Wissem, Bishnu et Kamrul pour les nombreuses réalisations de leurs thèses de science soutenues. Merci Bernard, Malika, Isaure, Hamed et Yasaman pour les conseils réguliers et les bons moments. Merci à tous les collègues du Loria. Merci à tous les collègues de Telecom Nancy et aux collègues administratifs et notamment à Antoinette, Isabelle, Carole pour me faciliter énormément la vie. Many thanks to UFC friends ! José, Joao Paulo !

Last but not least, je voudrais adresser mes remerciements les plus chaleureux à ma chère et formidable épouse Manel et à mes chers enfants : Adam, Youssef et Yanis : Vous êtes l'essence de ma vie.

Table des matières

Table des figures

Introduction

Chapitre 1

Introduction

1.1 Extraction de connaissances à partir des graphes

Les graphes sont utilisés dans de nombreux domaines, allant des réseaux sociaux à la sécurité informatique en passant par la géographie et la bioinformatique. Leur polyvalence et leur capacité à modéliser des données complexes en font un outil précieux pour la résolution de plusieurs problèmes. En règle générale, l'extraction de connaissances à partir des graphes consiste à utiliser des techniques d'analyse de graphes pour découvrir des connaissances cachées dans les données représentées par les graphes. Dans le contexte de nos travaux de recherche, nous avons fondé nos contributions sur le processus classique d'extraction de connaissances à partir de données (ECD) illustré dans la Figure 1.1 [Fayyad et al., 1996]. Il s'agit d'un processus en plusieurs étapes qui vise à extraire des connaissances utiles à partir des données brutes. Il débute par la considération de données hétérogènes, qui sont préparées (c'est-à-dire, intégrées, structurées), fouillées, pour générer des régularités qui sont ensuite interprétées pour donner naissance à des connaissances [Coulet, 2019]. Ce processus nécessite la compréhension approfondie des données, des techniques d'analyse et de la manière dont les résultats peuvent être interprétés et appliqués.

Dans le contexte de nos travaux de recherche, la représentation en graphes a été utilisée pour représenter des données biologiques et le processus d'ECD a été utilisé pour extraire des connaissances utiles qui sont validées et interprétées par des experts de l'équipe Capsid du Loria-Inria Nancy. Comment utiliser la représentation en graphes pour représenter des données biologiques et extraire des connaissances utiles tout en considérant la complexité et la quantité des données

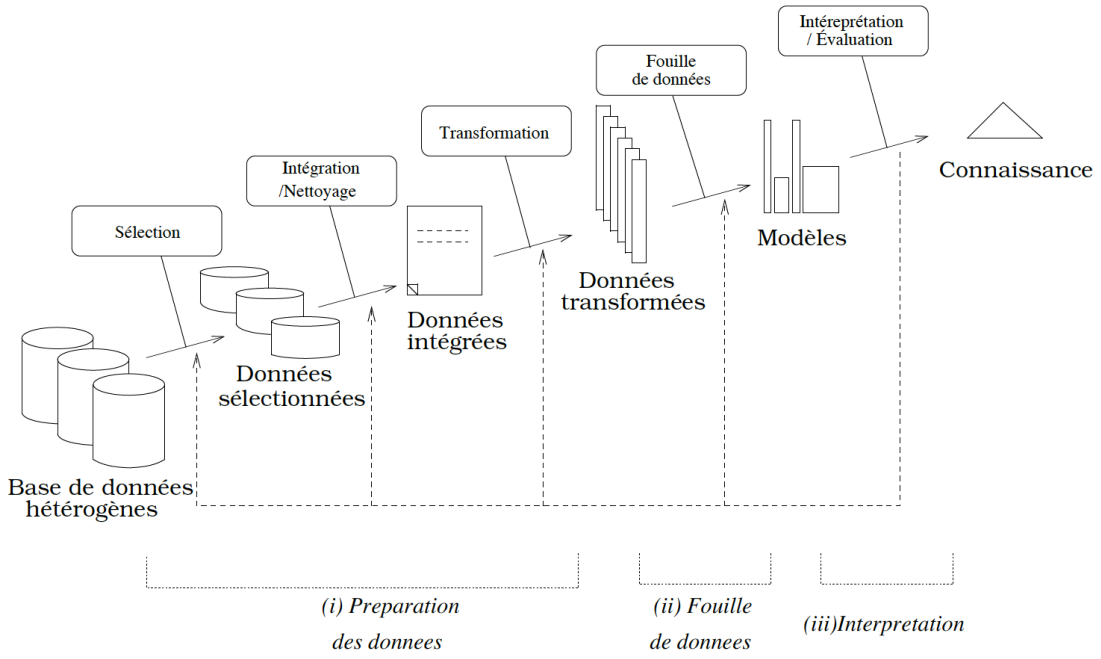


FIGURE 1.1 – Représentation du processus d'Extraction de Connaissances à partir des données [Fayyad et al., 1996].

disponibles est la problématique transversale de ce mémoire d'HDR et constitue un fil de lecture.

1.2 Contexte : projets de recherche et encadrements

Les contributions décrites dans ce mémoire d'HDR ont été réalisées dans le cadre de projets de recherche, ainsi que dans le cadre de l'encadrement de thèses et de Masters, auxquels j'ai été directement impliqué après l'obtention de mon doctorat.

Une description plus détaillée de mon profil est disponible sur mon curriculum vitae en ligne, que vous pouvez consulter à l'adresse `https://members.loria.fr/SAridhi/cv/`. J'ai choisi de séparer délibérément ces deux documents afin que ce mémoire d'HDR se focalise exclusivement sur l'aspect scientifique de mon projet d'habilitation.

La thèse de Manel Zoghlami [Zoghlami, 2019]. J'ai co-encadré Manel Zoghlami entre Septembre 2015 et Décembre 2019. Les travaux de cette thèse traitent de l'utilisation des approches d'apprentissage multi-instances (MI) pour la prédiction de la résistance aux rayonnements ionisants chez les bactéries [Zoghlami et al., 2020] [Aridhi et al., 2016b]. C'est à travers cet encadrement que je me suis intéressé à l'utilisation de l'intelligence artificielle (IA) pour résoudre des

problèmes biologiques.

La thèse de Bishnu Sarker [Sarker, 2021]. J'ai co-encadré Bishnu Sarker entre Novembre 2017 et Avril 2021. Cette thèse traite des approches à base de graphes pour effectuer l'annotation automatique des fonctions protéiques tout en étudiant l'impact de l'architecture en domaines sur les fonctions protéiques [Sarker et al., 2020b, Sarker et al., 2020a, Sarker et al., 2022].

La thèse de Wissem Inoubli [Inoubli, 2021]. J'ai co-encadré Wissem Inoubli entre Novembre 2017 et Janvier 2021. L'intérêt de Wissem Inoubli pour les méthodes de calcul distribué sur les graphes nous a menés à des travaux sur la classification non supervisée des grands graphes dynamiques qui ont été publiés dans des revues internationales [Inoubli et al., 2018] [Inoubli et al., 2022].

La thèse de Md Kamrul Islam [Islam, 2022]. J'ai co-encadré Kamrul Islam entre Novembre 2019 et Décembre 2022. Le travail de thèse de Kamrul s'intéresse à la prédiction de liens dans des graphes de connaissances et à l'application de ces méthodes pour la tâche de repositionnement de médicaments. Les travaux de cette thèse ont mené à des publications dans des conférences [Islam et al., 2021] et des revues internationales [Islam et al., 2022] [Islam et al., 2023].

Le projet Tempographs. J'ai été responsable scientifique et coordinateur du projet TempoGraphs [1], un projet de collaboration franco-brésilien PRC-CNRS-FAPs entre l'Université de Lorraine et l'Université de Céara au Brésil. Ce projet traite l'étude et l'application des outils d'apprentissage automatique et des graphes temporels pour l'annotation automatique des protéines. Ce projet a permis dans un premier temps d'étendre le travail de thèse de Wissem Inoubli pour le traitement des graphes temporels dans un contexte distribué [Inoubli et al., 2017]. Dans un deuxième temps, ce projet nous a permis d'étudier l'utilisation de différentes méthodes de similarité dans le contexte de travail de Bishnu Sarker [Veras et al., 2022].

1. `https://tempographs.loria.fr/`

1.3 Notions préliminaires

1.3.1 Les protéines et leurs annotations

Les protéines sont des macromolécules importantes qui constituent la base de la vie et jouent un rôle vital dans tout organisme vivant. Dans chacune des cellules qui composent un être vivant, les protéines remplissent de multiples fonctions dont la connaissance permet de mieux comprendre les mécanismes biologiques pour, entre autres, anticiper l'évolution des maladies et concevoir de nouveaux médicaments. Selon le dogme central de la biologie moléculaire, l'information génétique codée à l'intérieur de l'ADN est transcrite en ARN et à partir de l'ARN, elle est traduite en protéines qui agissent finalement à l'intérieur de la cellule. Les protéines sont, en quelque sorte, les produits finaux du processus de décodage de l'information génétique.

L'annotation fonctionnelle de protéines consiste à attribuer des caractéristiques fonctionnelles appropriées aux protéines. Ces caractéristiques sont souvent des termes de l'ontologie GO ou bien des numéros EC lorsqu'il s'agit d'une enzyme.

L'ontologie Gene Ontology (GO). Gene Ontology (GO) [Ashburner and et al., 2000] répertorie le vocabulaire contrôlé des termes qui décrivent les fonctions et ses termes font référence aux diverses fonctions biologiques que les protéines et les gènes remplissent dans les cellules et les organismes vivants. Dans GO, les termes fonctionnels sont présentés dans une hiérarchie en trois graphes acycliques dirigés (DAG), à savoir : 1) le processus biologique (BP), 2) la fonction moléculaire (MF) et 3) le composant cellulaire (CC). Chaque terme GO est un nœud dans le DAG. Chaque DAG commence à partir d'un terme racine et se connecte à d'autres termes hiérarchiques à travers différents types de liens indiquant différents types de relations. Les relations les plus utilisées sont « est un », « fait partie de » et « régule » tels qu'ils apparaissent dans la base de données GO.

Les numéros EC. La nomenclature EC (EC est l'abréviation de *Enzyme Commission*, la Commission des Enzymes) est une classification numérique des enzymes, basée sur la réaction chimique qu'elles catalysent [Cornish-Bowden, 2014]. Cette nomenclature attribue à chaque enzyme un numéro à quatre chiffres. Ce système de classification a une structure hiérarchique. Le premier niveau comprend six classes principales d'enzymes : (i) les oxydoréductases, (ii) les transférases, (iii) les hydrolases, (iv) les lyases, (v) les isomérases et (vi) les ligases, représentées

par le premier chiffre. Chaque nœud de classe principale plus loin s'étend à plusieurs nœuds de sous-classes, spécifiant des sous-classes d'enzymes, représentées par le deuxième chiffre. De même, le troisième chiffre indique la sous-sous-classe, et le quatrième chiffre désigne les sous-sous-sous-classes. Considérons comme exemple une enzyme de restriction de type II, qui est annotée EC 3.1.21.4. Le premier chiffre, 3, indique qu'il s'agit d'une hydrolase. Le deuxième chiffre, 1, indique qu'il agit sur les liaisons ester. Le troisième chiffre, 21, indique qu'elle est une endodésoxyribonucléase produisant des 5-phosphomonoesters. Le dernier chiffre, 4, spécifie qu'il s'agit d'une désoxyribonucléase site-spécifique de type II.

1.3.2 Les graphes : notations et définitions

Les graphes

Un **graphe**, $G = (V, E, X, W)$ est une représentation d'un réseau, où V est l'ensemble des sommets (ou nœuds), X est un ensemble d'attributs de nœud , $E \subseteq V \times V$ est un ensemble d'arêtes (ou de liens) qui relie deux nœuds et W est un ensemble d'attributs de lien. Les nœuds représentent des entités dans des réseaux, telles que des personnes dans un réseau social, des protéines dans un réseau d'interactions protéine-protéine (PPI), des auteurs et des articles dans un réseau de coauteurs, etc. Les nœuds peuvent contenir des informations descriptives, telles que des noms, des types, des sources. Les liens représentent des connexions entre des paires d'entités, telles que le co-auteur dans les réseaux d'auteurs, l'amitié dans les réseaux sociaux, l'interaction dans les réseaux PPI. Les liens peuvent contenir des informations descriptives, telles que des noms, des sources, des pondérations.

$|V|$ et $|E|$ indique respectivement le nombre de nœuds et le nombre de liens. Le lien entre deux nœuds $x \in V$ et $y \in V$ est noté $e_{xy} \in E$. e_{xy} est appelé un lien dirigé si le sens du lien va du nœud x au nœud y et un lien non dirigé sinon. Pour un lien dirigé, $e_{xy} \in E$, x et y sont respectivement appelés source et destination. Deux nœuds $x, y \in V$ sont **voisins** s'ils sont liés, *c'est-à-dire*, $e_{xy} \in E$. Les voisins du nœud x forment collectivement un ensemble de voisins Γ_x du nœud. Le nombre de voisins pour un nœud, x est appelé **degré** du nœud et noté $|\Gamma_x|$. Une **matrice d'adjacence**, A est une représentation matricielle du graphe, G où un élément A_{xy} indique si les nœuds x et y sont adjacents ou non dans le graphe.

$$A_{xy} = \begin{cases} 1 & \text{si } e_{xy} \in E \\ 0 & \text{si } e_{xy} \notin E \end{cases}$$

Un **chemin**, $P_{x,z}$ est une séquence de liens $e_{xy} - e_{ya} - .... - e_{tz}$ entre les nœuds $x, z \in V$ où $x, y, a, ..., t, z \in V$ et $e_{xy}, e_{ya}, ..., e_{tz} \in E$. Le nombre de liens dans $P_{x,z}$ est appelé **longueur du chemin**. Un chemin de longueur l entre deux nœuds x et y est noté $P^{l}_{x,y}$. Il peut exister plusieurs chemins dans G entre une paire de nœuds x, z. Parmi les chemins, le chemin avec la plus petite longueur est appelé le plus court chemin noté $P^{shortest}_{x,z}$. Un graphe est connexe s'il existe un ou plusieurs chemins entre chaque paire de nœuds.

Si E ne contient que des liens orientés alors G est appelé un **graphe orienté**. D'autre part, tous les liens dans E sont non orientés dans un **graphe non orienté**. Un graphe est appelé **graphe pondéré** si pour chaque lien, un poids est attribué. Sinon, le graphe est appelé un **graphe non pondéré**. Le poids d'un lien peut représenter un coût, une longueur ou une capacité. Un graphe est appelé **multi-graphe** s'il autorise plusieurs liens entre une paire de nœuds.

Clustering structurel de graphes

Le clustering de graphes consiste à découper un graphe en plusieurs partitions ou sous-graphes. Comme avec d'autres techniques de clustering, nous devons utiliser une ou plusieurs métriques pour mesurer la similarité entre deux sommets ou partitions dans le graphe. Dans la technique de clustering structurel, la structure ou la topologie du graphe est divisée en un ensemble de sous-graphes relativement distants et un ensemble de sommets fortement connectés. L'algorithme de référence de clustering structurel est l'algorithme *SCAN* [Xu et al., 2007] qui utilise la similarité structurelle entre les sommets pour effectuer le regroupement.

Dans ce qui suit, nous présentons quelques définitions essentielles du clustering structurel de graphes, qui seront principalement utilisées dans la partie II du mémoire.

Définition 1.3.1. *(Voisinage structurel) Le voisinage structurel d'un sommet v, est noté $N(v)$,*

et représente tous les voisins d'un sommet donné $v \in V$, y compris le sommet v :

$$N(v) = \{u \in V | (v, u) \in E\} \cup \{v\} \tag{1.1}$$

Définition 1.3.2. *(Similarité structurelle) La similarité structurelle entre chaque paire de sommets (u, v) dans un ensemble d'arêtes E représente un certain nombre de voisins structurels partagés entre u et v. Il est défini par $\sigma(u, v)$.*

$$\sigma(u, v) = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)| \,.\, |N(v)|}} \tag{1.2}$$

Après avoir calculé la similarité structurelle avec l'équation 1.2, l'algorithme de référence *SCAN* utilise deux paramètres ϵ et μ pour détecter les sommets noyaux dans un graphe donné G.

Définition 1.3.3. *(ϵ-voisinage) Chaque sommet a un ensemble de voisins structurels, comme il est mentionné dans la définition 1.3.1. Pour regrouper un sommet et ses voisins dans le même cluster, ils doivent avoir une connexion forte (notée ϵ-voisinage) entre eux. SCAN utilise un seuil ϵ et l'équation 1.3 pour filtrer, pour chaque sommet, ses connexions les plus fortes. Le voisinage ϵ est défini comme suit.*

$$N_\epsilon(u) = \{N(u) | \sigma(u, v) \geq \epsilon\} \tag{1.3}$$

Le seuil ϵ $(0 < \epsilon \leqslant 1)$ montre dans quelle mesure deux sommets u et v sont connectés en fonction des voisins structurels partagés. De plus, il représente une métrique avec laquelle les sommets les plus importants, également appelés sommets *noyaux*, sont détectés.

Définition 1.3.4. *(Sommet noyau ou Core) La détection des sommets noyaux est une étape fondamentale de l'algorithme SCAN. Elle consiste à trouver les sommets dominants dans un graphe donné G. Cette étape permet de construire l'ensemble des clusters. Un sommet noyau v est un sommet qui a un nombre suffisant de voisins fortement connectés avec lui $N_\epsilon(v)$. Nous utilisons μ comme nombre minimum de voisins fortement connectés (voir Définition 1.3.3). L'ensemble des sommets noyau est modélisé comme suit :*

$$V_c = \{v \mid |N_\epsilon(v)| \geq \mu\} \tag{1.4}$$

Définition 1.3.5. *(Sommet bordure ou Border) Soit v_c un sommet noyau. v_c a deux listes de voisins structurels : (1) voisins faiblement connectés à v_c, également appelés sommets bruits ($N(V_c) \setminus N_\epsilon(V_c)$), et (2) voisins fortement connectés appelés voisins structurels accessibles. Dans notre travail, les voisins structurels accessibles sont appelés sommets bordures. $N_\epsilon(V_c)$ représente l'ensemble des sommets de bordures d'un sommet noyau V_c.*

Une fois les nœuds et leurs sommets de bordures déterminés, il est simple de démarrer une étape de clustering. Pour ce faire, nous utilisons la définition suivante :

Définition 1.3.6. *(Groupe ou Cluster) Un cluster C ($|C| \geq 1$) est un sous-ensemble non vide de sommets composé de l'ensemble des sommets noyaux et leurs sommets de bordures.*

L'algorithme de construction des clusters se compose des principales étapes suivantes :

1. Choix aléatoire d'un sommet noyau dans la liste des noyaux et création d'un cluster C.
2. Ajout du noyau et de ses bordures dans le même cluster. En parallèle, l'algorithme vérifie si la liste des nœuds frontières contient un sommet noyau.
3. Ajout récursif des bordures dans le même cluster, en appliquant cette étape jusqu'à ce que toutes les bordures des noyaux connectés soient ajoutées.
4. Choix d'autres sommets noyaux et exécution des étapes précédentes jusqu'à ce que tous les sommets noyaux soient vérifiés.

Parmi les informations fondamentales renvoyées par *SCAN*, par rapport aux autres algorithmes de clustering de graphes, nous mentionnons les informations sur les sommets de type pont et les *outliers*, définis comme suit :

Définition 1.3.7. *(Pont ou Bridge) Un pont b est un sommet $v \in V$ qui n'appartient à aucun cluster après avoir effectué l'étape de clustering, et qui a au moins deux liens (arêtes) avec deux clusters différents.*

Définition 1.3.8. *(Outlier) Un sommet outlier o est un sommet $v \in V$ qui n'appartient à aucun cluster après l'étape de clustering, et ce n'est pas un pont.*

L'étape de clustering agrège les sommets noyaux et leurs bordures en un ensemble de clusters. Cependant, certains sommets n'ont pas de connexions fortes avec un sommet noyau, ce qui ne

donne pas la possibilité de rejoindre un cluster. Dans ce contexte, l'algorithme *SCAN* classe ces sommets en deux familles : les ponts et les *outliers*. Un sommet v, qui ne fait partie d'aucun cluster et qui a au moins deux voisins dans des clusters différents, est appelé pont. Sinon, il est considéré comme *outlier*.

Les graphes de connaissances

Un graphe de connaissances (KG) est un graphe qui permet d'encoder plus d'informations sémantiques par rapport à un graphe simple. Les KG sont généralement représentés à l'aide du modèle de données RDF (Resource Description Framework) [Klyne, 2004]. Le modèle RDF permet l'occurrence de liens multiples, de boucles, de liens dirigés, de noms de liens obligatoires et d'attributs de nœud et de lien auxiliaires [Paulheim, 2017]. Formellement, un graphe de connaissances est défini comme $KG = (V, R, Q)$ [Gao et al., 2022], où

— V est un ensemble de nœuds représentant des entités ;

— R est un ensemble de noms de relations ;

— $Q \subseteq V \times R \times V$ est un ensemble de faits (ou triplets positifs) ;

Les faits sont formatés sous la forme de triplet, $(h, r, t) \in Q$ où $h \in V$ et $t \in V$ qui sont respectivement appelés les entités de tête et de queue et $r \in R$ est une relation dirigée de h vers t.

Sur la base des propriétés structurelles, une relation peut être symétrique ou anti-symétrique, inverse d'une autre relation et composée de plusieurs relations [Sun et al., 2018]. Formellement, si $\{(h, t)\}$ est l'ensemble des paires d'entités pour une relation r, alors la relation $r \in R$ est

— anti-symétrique, si $\forall_{(h,t)}(h, r, t) \in Q \implies (t, r, h) \notin Q$.

— symétrique, si $\forall_{h,t}(h, r, t) \in Q \implies (t, r, h) \in Q$.

— composée de deux relations, $r_1,\ r_2 \in R$, si $\forall_{h,t}(h, r_1, z) \in Q \wedge (z, r_2, t) \in Q \implies (h, r, t) \in Q$.

— inverse d'une autre relation, $r_1 \in R$, si $\forall_{h,t}\{(h, r, t) \in Q \implies (t, r_1, h) \in Q\}$.

— un à plusieurs, si $(h, r, t_1) \in Q \wedge (h, r, t_2) \in Q \wedge .... \wedge (h, r, t_n) \in Q$.

— plusieurs à un, si $(h_1, r, t) \in Q \wedge (h_2, r, t) \in Q \wedge .... \wedge (h_n, r, t) \in Q$.

Plongement de graphes de connaissances

L'idée principale du plongement de graphes de connaissances (ou *Knowledge Graph Embedding (KGE)*) est de représenter les entités et les relations d'un graphe de connaissances dans des espaces vectoriels. Cela permet de simplifier la manipulation du graphe tout en préservant l'essentiel de sa sémantique. L'objectif principal est de vérifier que les éléments qui sont proches dans le graphe de connaissances se trouvent également proches dans l'espace vectoriel où ils sont plongés. En général, les modèles de KGE emploient les techniques relatives au domaine d'apprentissage automatique, et se fondent sur des fonctions objectives précises.

Prédiction de liens dans des graphes de connaissances

Étant donné un graphe de connaissances $KG = (V, R, Q)$, où V est l'ensemble des entités, R est l'ensemble des noms de relations, Q est l'ensemble des faits, le problème de prédiction de lien est défini comme la prédiction de l'entité manquante dans un triplet $(h, r, ?)/(?, r, t)$, où ? représente l'entité manquante [Rossi et al., 2021, Liang et al., 2022].

Les performances de la tâche de prédiction de lien sont définies avec deux métriques largement utilisées : Hit@z et le Mean Reciprocal Rank (MRR) [Wang et al., 2021]. Les métriques sont définies en fonction du rang du triplet positif de test. Hit@z est défini comme le nombre moyen de fois qu'un triplet de test positif figure parmi les z triplets les mieux classés ; alors que MRR est le rang réciproque moyen du triplet de test positif [Bordes et al., 2013, Yang et al., 2015]. La plage des deux scores est de 0 à 1. La valeur la plus élevée de MRR démontre le meilleur classement des triplets de test positifs et un meilleur classement offre de meilleures performances de prédiction. De plus, un score Hit@z plus élevé indique de meilleures performances. Pour illustrer, considérons le triplet de test positif $q = (h, r, t) \in \mathbb{D}$. Un ensemble de triplets négatifs $q'_t = \{(h, r, t') \notin \mathbb{Q} | t' \in \mathbb{E}\}$ est généré par simple perturbation de triplet (remplacement de la queue par d'autres entités) [Bordes et al., 2013, Ji et al., 2015] confirmant qu'aucun triplet positif n'existe dans q'_t. Les triplets de $q \cup q'_t$ sont ensuite classés par ordre décroissant de leurs scores (calculés par la fonction de score de l'approche de prédiction). Le rang du triplet de test positif q dans $q \cup q'_t$ est défini comme $rank^t_q$. Sur la base du rang de chaque triplet de test positif q, les métriques de performance sont définies dans les équations 1.5 et 1.6.

$$Hit^t@z = \frac{1}{|\mathbb{D}|}\sum_{q\in\mathbb{D}} hit_q^t, \quad hit_q^t = \begin{cases} 1, & if\ rank_q^t \leq z \\ 0, & \text{otherwise} \end{cases} \tag{1.5}$$

$$MRR^t = \frac{1}{|\mathbb{D}|}\sum_{q\in\mathbb{D}} \frac{1}{rank_q^t} \tag{1.6}$$

L'ensemble du processus d'évaluation est également répété en corrompant également l'entité principale de chaque triplet positif et $Hit^h@z$, MRR^h sont calculés. Les métriques finales $Hit@z$, MRR sont la moyenne des métriques tête et queue *c'est-à-dire*, $Hit@z = (Hit^t@z + Hit^h@z)/2$ et $MRR = (MRR^t + MRR^h)/2$. En suivant la majorité des travaux de recherche sur la prédiction de liens dans des graphes de connaissances, nous considérons $z \in \{1, 3, 10\}$.

1.4 Organisation du manuscript

Ce manuscrit d'HDR a pour but de présenter mes travaux (réflexifs et expérimentaux) et contributions qui illustrent le mieux l'utilisation efficace de la représentation en graphe pour la découverte des connaissances biologiques à grande échelle.

Les parties I et II de ce mémoire d'HDR présentent éssentiellement mes activités de recherche. Ces parties sont constituées de quatre chapitres principaux.

Les deux premiers chapitres 2 et 3 traitent en particulier les contributions d'apprentissage automatique sur des graphes et leurs applications pour résoudre deux problèmes biologiques : (1) l'annotation fonctionnelle de protéines et (2) le repositionnement de médicaments. Le chapitre 2 explique et met en évidence l'intérêt de la décomposition en domaines des protéines pour représenter les protéines par des graphes de similarité. Il présente également l'application de méthodes de propagation d'étiquettes sur ces graphes pour l'annotation fonctionnelle de protéines. Le chapitre 3 se concentre sur la prédiction de liens dans des graphes de connaissances biologiques pour le repositionnement de médicaments. Il décrit notamment notre travail sur la génération d'exemples négatifs et l'utilisation de techniques de plongement de graphes de connaissances pour améliorer la prédiction de liens.

Le chapitre 4 traite la proposition d'approches distribuées pour l'analyse de grands graphes.

Nous décrivons dans un premier temps une étude comparative sur les plateformes de traitement et d'analyse de données massives et plus particulièrement des grands graphes. Dans un second temps, nous décrivons notre travail sur le clustering de grands graphes dynamiques.

Le chapitre 5 est consacré à la description de nos travaux sur le passage à l'échelle des approches de plongement dans des grands graphes de connaissances.

La partie III de ce document explicite mes perspectives de recherche dans le contexte de l'utilisation des graphes simples ou de connaissances et des techniques d'apprentissage automatique de pointe pour résoudre des problèmes biologiques.

Première partie

Intelligence artificielle et graphes de connaissances pour l'annotation de proteines et le repositionnement de médicaments

Chapitre 2

Propagation de labels dans des graphes de protéines

2.1 Introduction

Il est important de noter qu'une quantité importante de séquences protéiques sont déjà disponibles dans les bases de données publiques. Par exemple, la base de données « UniProt KnowledgeBase » (UniProtKB) [2] est la plus grande base de données publique pour le stockage des séquences protéiques. UniProtKB contient plus de 250 millions de séquences selon la version de Février 2023. Ce grand volume de séquences protéiques ouvre, d'une part, plusieurs opportunités pour réaliser des analyses utiles et pour répondre à des questions de recherche en biologie. D'autre part, cela pose des défis en raison du fait que cette énorme base de données est presque impossible à annoter manuellement. UniProtKB est divisé en deux parties : 1) UniProtKB/SwissProt et 2) UniProtKB/TrEMBL. Dans UniProtKB/SwissProt, les séquences protéiques sont annotées manuellement. Ce travail d'annotation manuelle nécessite un travail important pour les annotateurs humains. En fait, il faut beaucoup de temps pour lire des publications, trouver des informations sur une protéine particulière, identifier ses propriétés et produire finalement une annotation. Ce processus est donc très coûteux. Ce sont les principales raisons de la croissance très lente d'UniProtKB/SwissProt au cours des années. Selon la version 2023 de UniProtKB/SwissProt, il existe environ 569,000 séquences de protéines dont les annotations ont été produites et examinées

2. `https://www.uniprot.org/`

manuellement. Au contraire, dans UniProtKB/TrEMBL, les séquences protéiques reçoivent des annotations générées par des outils informatiques et qui ne sont pas revues manuellement. Quand UniProtKB reçoit une nouvelle séquence protéique, elle est ajoutée à UniProtKB/TrEMBL et la séquence est disponible en ligne pour une enquête plus approfondie. Ainsi, UniProtKB/TREMBL a connu une très forte croissance au fil des ans, notamment par le moyen de la traduction automatique des régions codantes des génomes au fur et à mesure de leur séquençage. Selon la version de Février 2023, UniProtKB/TREMBL contient environ 249 millions de séquences protéiques sans annotation fonctionnelle validée manuellement. Pour enrichir et exploiter cette quantité importante de séquences protéiques, il est fortement souhaité de les annoter avec des propriétés fonctionnelles telles que les numéros de commission enzymatique (EC) ou les termes de l'ontologie Gene Ontology (GO) décrivant leurs fonctions. Ce travail est considérable et, pour réduire l'écart entre le volume des séquences protéiques non annotées et celui des séquences annotées, il est essentiel de développer des techniques d'annotation automatique de la fonction des protéines. Dans ce chapitre, je présente nos contributions majeures pour l'annotation automatique des fonctions protéiques en utilisant la représentation en graphe et les méthodes de propagation des étiquettes (ou *labels*) dans les graphes.

2.2 Des approches à base de graphes pour l'annotation

Dans le cadre de la thèse de Bishnu Sarker [Sarker, 2021], nous avons travaillé sur la proposition d'approches à base de graphes pour l'annotation des protéines. Nous avons principalement proposé *GrAPFI*("Graph-based Automatic Protein Function Inference") [Sarker et al., 2020b] pour l'annotation automatique des protéines par les numéros EC et par des termes de l'ontologie GO. La méthode *GrAPFI* explore la composition en domaines des protéines en construisant un graphe de similarité de domaines et en effectuant une propagation d'étiquettes basée sur le voisinage pour prédire les annotations des protéines non annotées.

Une vue générale de la méthode *GrAPFI* est présentée dans la Figure 2.1.

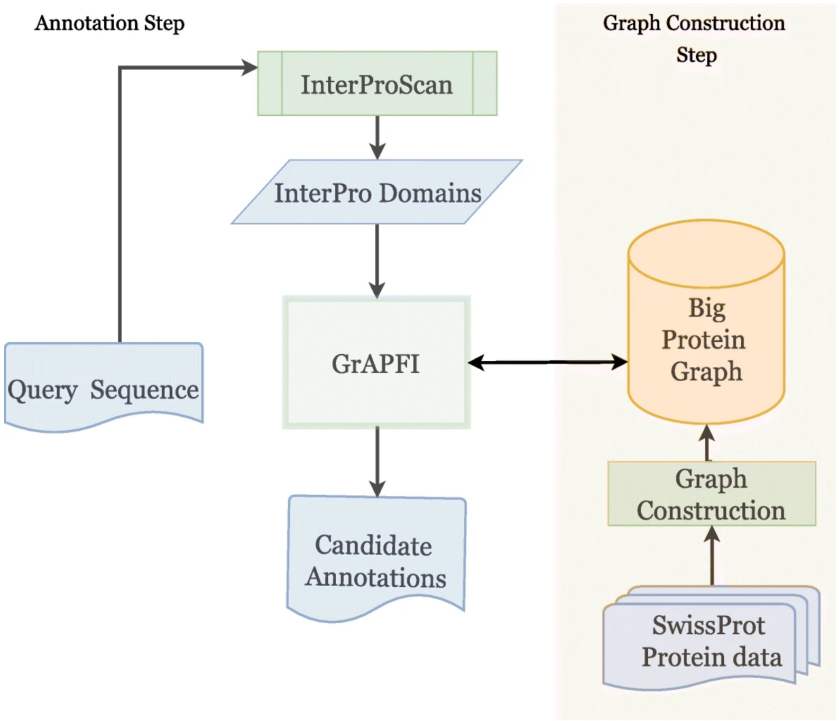


FIGURE 2.1 – Le workflow d'annotation utilisé dans *GrAPFI*. La partie droite du flux de travail représente la construction du graphe à l'aide de protéines validées de l'UniprotKB/Swissprot. La partie gauche montre le processus d'annotations. [Sarker et al., 2020b]

2.2.1 Construction d'un graphe de protéines

Nous présentons ici une nouvelle façon de connecter des séquences de protéines en utilisant leurs domaines associés. Les domaines peuvent être considérés comme des blocs de construction naturels des protéines. En raison de l'évolution, les domaines protéiques peuvent avoir subis des changements tels que la duplication, la fusion, la recombinaison pour produire des protéines avec des structures et des fonctions distinctes [Kummerfeld and Teichmann, 2009]. Ici, chaque nœud du graphe représente une protéine, tandis qu'un lien entre deux nœuds signifie que les protéines présentent un niveau donné de similarité de domaine. Ainsi, chaque nœud u est identifié par un ensemble d'étiquettes $L(u)$, a un ensemble de voisins $N(u)$, et pour chaque voisin v, un poids associé Wu, v. L'objectif général est de propager des étiquettes (c'est-à-dire des annotations) à partir de nœuds ayant des étiquettes vers des nœuds similaires dépourvus d'étiquettes.

Pour illustrer la construction du graphe des protéines, prenons l'exemple de cinq protéines désignées par les noms symboliques P1, P2, P3, P4 et P5. Supposons que ces protéines soient com-

posées des domaines suivants : $D1 = \{d1, d2, d3, d4\}$,$D2 = \{d1, d3, d5\}$,$D3 = \{d1, d2, d10\}$,$D4 = \{d5, d6, d1\}$, et $D5 = \{d4, d1, d10, d40, d7, d9, d12, d52, d100\}$.

La composition des domaines d'une protéine est l'ensemble des domaines trouvés dans une séquence protéique et considérés indépendamment de l'ordre d'apparition dans la séquence. Par exemple, les informations de domaine dans $D1 = \{d1, d2, d3, d4\}$ peuvent être utilisées dans n'importe quel autre ordre. Par conséquent, la composition n'est pas strictement linéaire. Le chevauchement des domaines n'est pas pris en compte tant que les domaines chevauchants ont une nouvelle identification de domaine. Il est alors évident que les protéines P1 et P2 possèdent deux domaines en commun (d1 et d3). Par conséquent, les protéines P1 et P2 peuvent être liées et le nombre de domaines partagés peut servir de poids de liaison donné par $W_{P1,P2} = |\{d1, d2, d3, d4\} \cap \{d1, d3, d5\}| = |\{d1, d3\}| = 2$.

De la même manière, les protéines P1 et P5 peuvent être liées avec un poids de liaison de $|\{d1, d2, d3, d4\} \cap \{d4, d1, d10, d40, d7, d9, d12, d52, d100\}| = |\{d1, d4\}| = 2$. Dans les deux cas, le poids du lien est de 2. Cependant, le poids du lien calculé de cette manière ne reflète pas la force relative de la relation entre les protéines. Plus précisément, dans le premier cas, les deux protéines ont $|\{d1, d2, d3, d4) \cup \{d1, d3, d5\}| = |\{d1, d2, d3, d4, d5\}| = 5$ domaines différents, dont deux sont partagés. Dans le second cas, il y a $|\{d1, d2, d3, d4\} \cup \{d4, d1, d10, d40, d7, d9, d12, d52, d100\}| = 11$ domaines différents dont encore deux sont partagés. Bien que deux domaines soient partagés dans chaque cas, la protéine P1 est intuitivement plus alignée avec P2 que P5. Par conséquent, au lieu d'utiliser le score de similarité brut ci-dessus, nous avons utilisé l'indice de similarité de Jaccard, ou coefficient de similarité de Jaccard, pour mieux refléter la similarité dans la composition.

Ceci est calculé comme $\frac{|A \cap B|}{|A \cup B|}$, où A et B sont les deux ensembles de domaines constitutifs. À l'aide de l'indice de similarité Jaccard, les poids des liens pour P1 et P2 sont calculés comme suit : $W_{P1,P2} = \frac{|\{d1,d2,d3,d4) \cap \{d1,d3,d5\}|}{|\{d1,d2,d3,d4\} \cup \{d1,d3,d5\}|} = \frac{|\{d1,d3\}|}{|\{d1,d2,d3,d4,d5\}|} = \frac{2}{5} = 0.4$.

De même, pour P1 et P5, le poids du lien est calculé comme suit : $W_{P1,P5} = \frac{|\{d1, d2, d3, d4\} \cap \{d4, d1, d10, d40, d7, d9, d12, d52, d100\}|}{|\{d1, d2, d3, d4\} \cup \{d4, d1, d10, d40, d7, d9, d12, d52, d100\}|} = \frac{2}{11} = 0.18.$

En utilisant l'indice de similarité Jaccard, le graphe final est construit en deux étapes simples. Dans la première étape, les informations sur les protéines examinées sont analysées pour collecter les domaines constitutifs de chaque protéine. Si les données d'apprentissage ne contiennent que des séquences, l'outil InterProScan [Jones et al., 2014] est appliqué à chacune des séquences pro-

téiques pour trouver les domaines Interpro associés. Interpro est une base de données intégrées de domaines protéiques utilisée pour la classification et l'annotation automatique de protéines. Ensuite, le graphe est construit en utilisant la composition en domaines des protéines.

Il convient de mentionner que l'ordre des domaines n'est pas maintenu lors du calcul de l'indice de similarité Jaccard. La composition de domaine pour chaque protéine contient l'ensemble des signatures InterPro uniques trouvées dans la séquence.

2.2.2 Annotation fonctionnelle de protéines par des numéros EC

Une fois le graphe construit à partir des protéines examinées, il peut être utilisé pour l'annotation de nouvelles séquences de protéines. Un algorithme de propagation d'étiquettes basé sur le voisinage est conçu pour effectuer la tâche d'annotation. Étant donné les domaines constitutifs d'une séquence de protéines d'entrée, toutes ses protéines voisines et leurs annotations sont extraites du graphe. Une fois les voisins obtenus, la fréquence pondérée des étiquettes est calculée à l'aide de la formule suivante : $f_u^i = \frac{\sum_{v \in N(u)} W_{u,v} \delta(v^i, i)}{\sum_{v \in N(u)} W_{u,v}}$, où f_u^i est le score pondéré de la fonction candidate i pour la protéine de requête u et $\delta(v^i, i)$ vaut 1 si la fonction v^i de la protéine v est identique à la fonction i, sinon 0. Les détails de l'algorithme de propagation de l'étiquette sont décrits dans l'algorithme 1.

Explicitement, pour une séquence d'entrée donnée, l'annotation fonctionne selon l'organigramme illustré dans la Figure 2.1.

Considérons une protéine requête P avec un ensemble de domaines D=(d5,d6,d101). Notre objectif est d'annoter cette protéine avec un numéro EC en suivant l'algorithme de propagation de l'étiquette, comme illustré dans la Figure 2.2.

Sur la base de la similarité de domaine, la protéine P aura une connexion avec les protéines P2 et P4 dans le graphe d'exemple en cours d'exécution. Les lignes pointillées montrent les liens de P à P2 et P4 dans le graphe avec les poids associés. Par conséquent, la protéine P aura P2 et P4 comme voisins. Après avoir trouvé les voisins, les annotations fonctionnelles de tous les voisins sont propagées avec les poids correspondants. Toutes les annotations fonctionnelles sont classées en fonction de leurs poids cumulés. La fonction la mieux classée est sélectionnée comme la meilleure annotation fonctionnelle pour la protéine P. Dans cet exemple, les annotations pondérées pour P sont EC3, EC5, EC6, EC1, EC2 avec des poids cumulés de 0,70, 0,50, 0,50,

Algorithm 1 Label Propagation in a protein graph

Input : A weighted undirected protein graph (Section 2.2), $G = (V, E)$, Minimum Jaccard Similarity Index, θ, and a query protein u with domain composition d
Output : Weighted EC Annotations
procedure LABEL PROPAGATION
    $Annotations \leftarrow \emptyset$
    $N' \leftarrow FilterNeighbors(N(u), \theta)$
    $ECs \leftarrow$ list of distinct ECs present among the neighbors N' **for** *each* $i \in ECs$ **do**
        $f_u^i = \frac{\sum_{v \in N'} W_{u,v} \delta(v^i, i)}{\sum_{v \in N'} W_{u,v}}$
    $Annotations \leftarrow Annotations \cup \{f_u^i\}$

    Rank the *Annotations*
    Select the top ranked annotations and assign it to the protein u
    end Procedure

function $FilterNeighbors\ (N(u), \theta))$
    $N' \leftarrow \emptyset$ **for** *each* $v \in N(u)$ **do**
        $W_{u,v} >= \theta$
    $N' \leftarrow N' \cup \{v\}$

    end if

    end for
    return N'

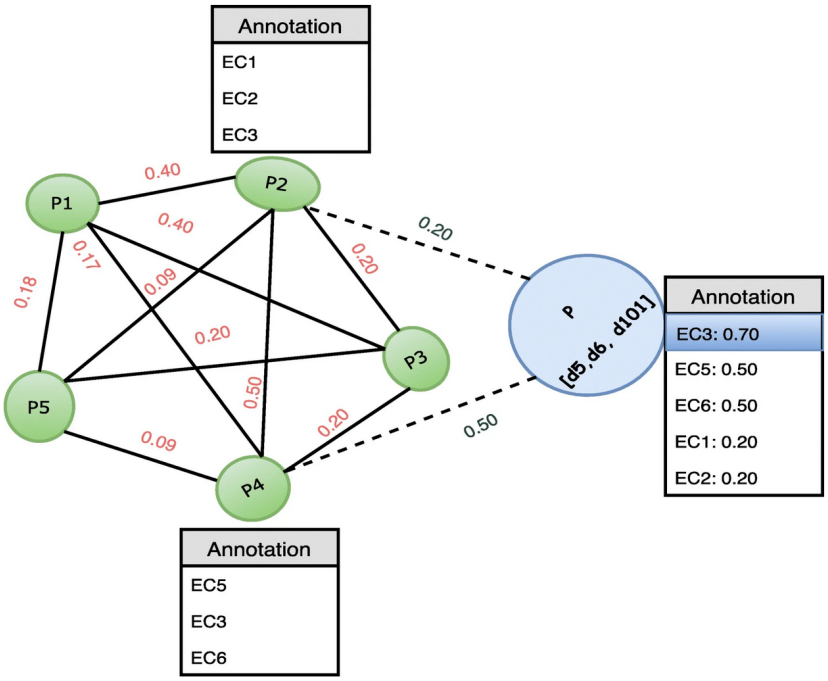


FIGURE 2.2 – Exemple d'annotation par des numéros EC. [Sarker et al., 2020b]

0,20 et 0,20, respectivement. Par conséquent, l'annotation fonctionnelle pour la protéine P est EC3 car elle a le poids le plus élevé parmi les marqueurs propagés. Il est clair qu'il est possible de sélectionner plus d'une annotation fonctionnelle à score élevé si l'on souhaite proposer plus d'une annotation candidate. De plus, les nœuds voisins pourraient être sélectionnés d'autres manières pour refléter les exigences du problème à résoudre.

2.2.3 Annotation fonctionnelle de protéines par des termes GO

GrAPFI a été initialement proposé pour la prédiction de la fonction des protéines par des numéros EC. Dans la suite de la thèse de Bishnu Sarker [Sarker, 2021], nous avons étendu et adapté la méthode *GrAPFI* pour l'annotation automatique des protéines avec des termes d'ontologie génique (GO) en la renommant *GrAPFI-GO* [Sarker et al., 2022]. *GrAPFI-GO* explore divers types de mesures de similarité basées sur des voisins communs dans le graphe de protéines construit par *GrAPFI*. De plus, les termes GO sont hiérarchisés selon les relations sémantiques parent-enfant. Par conséquent, nous proposons une technique d'élagage et de post-traitement efficace qui intègre à la fois la similarité sémantique et les relations hiérarchiques entre les termes GO. *GrAPFI-GO* effectue l'annotation par des termes GO à l'aide de *GrAPFI* combinée à un

post-traitement hiérarchique pour tirer parti de l'organisation hiérarchique de l'ontologie GO. Essentiellement, il y a deux parties dans le pipeline : (1) prédire l'ensemble des termes GO à l'aide de *GrAPFI*, et (2) élaguer l'ensemble d'annotations à l'aide de la similarité sémantique et du post-traitement hiérarchique.

Ainsi, le principe de propagation s'illustre comme suit : pour une protéine requête, nous cherchons toutes ses protéines voisines et leurs annotations dans le graphe pondéré en se basant sur sa composition en domaines. Après avoir obtenu les voisins, toutes les étiquettes des protéines voisines sont pondérées avec le poids des liens que ces voisins présentent avec la protéine de requête. Lors de la récupération des voisins, il est possible de sélectionner uniquement les voisins qui répondent à un certain seuil de similarité. Cela signifie que les liens peuvent être filtrés en fonction d'un poids de coupure prédéfini. Pour chaque annotation candidate, *GrAPFI-GO* fournit un score de confiance, nommé score du modèle (MS), qui est calculé ainsi :

$$MS(u,i) = \frac{\sum_{v \in N(u)} W_{u,v} \sum_{j \in L(v)} \delta(j,i)}{\sum_{v \in N(u)} W_{u,v}} \tag{2.1}$$

où $MS(u,i)$ est le score pondéré de la fonction candidate i pour la protéine de requête u. Et $\delta(j,i)$ vaut 1 si la fonction j de la protéine v est la même que la fonction candidate i, sinon 0.

Pour illustrer le processus d'annotation de *GrAPFI-GO*, prenons une protéine P non annotée avec un ensemble de domaines $D = (d5, d6, d101)$. Selon la similarité de Jaccard, P est connecté avec P2 et P4 dans le graphe de similarité de domaine. Par conséquent, la protéine P aura P2 et P4 comme voisins, c'est-à-dire $N(P) = P2, P4$. Dès que $N(P)$ est calculé, les annotations GO des protéines voisines sont propagées avec les poids correspondants calculés à l'aide de l'équation. 2.1. Lors de la propagation de l'étiquette, nous ne comptons pas les protéines non annotées qui apparaissent dans N(p). Toutes les annotations fonctionnelles sont classées en fonction de leurs poids cumulés. Les annotations les mieux classées sont sélectionnées comme les meilleures annotations fonctionnelles pour la protéine P.

Il est important de mentionner que *GrAPFI-GO* ainsi que la plupart des outils d'annotation par des termes GO [Radivojac and et al., 2013] [Jiang and et al., 2016] produisent un grand nombre de prédictions. En raison de ce grand nombre d'annotations prédites pour chaque protéine, la précision du modèle diminue tandis que le rappel augmente. Cependant, les résultats

de ces approches soulèvent une grande inquiétude sur les faux positifs dans les prédictions. Par conséquent, nous avons besoin d'une méthode qui améliore la précision du modèle, et donc diminue les faux positifs dans l'ensemble prédit. Pour résoudre ce problème, nous avons adopté une technique d'élagage naïf en identifiant et en éliminant les annotations aberrantes à l'aide de la similarité sémantique. La mesure de la similarité sémantique entre les termes GO a toujours été une étape essentielle dans la recherche en bioinformatique fonctionnelle. Dans un ensemble d'annotations GO prédites pour une protéine, la similarité sémantique par paires entre les termes GO peut montrer à quel point ces termes sont liés les uns aux autres et pas seulement à la protéine. Nous avons utilisé la méthode GOGO [Zhao and Wang, 2018] pour calculer le score de similarité fonctionnelle (SS) entre les termes GO et l'avons donc utilisé pour calculer le score d'appartenance de chaque terme GO prédit.

2.3 Résultats

2.3.1 Préparation de données pour la prédiction des numéros EC

Nous avons collecté plus de 262 milles protéines à partir de la version de mars 2018 de la base de données Uniprot-KB/SwissProt [Consortium, 2015] en utilisant les règles suivantes : (1) chaque protéine doit contenir au moins une signature InterPro et (2) elle doit être annotée avec au moins une annotation EC. Après avoir obtenu les données protéiques de chacune des protéines, nous avons extrait la composition en domaines InterPro et les annotations EC. Ensuite, nous avons construit le graphe de protéines comme décrit dans la Section 2.3.3. Chaque protéine forme son propre nœud dans le graphe. Nous n'avons pas prétraité les données d'apprentissage pour supprimer la redondance. Au contraire, lors de l'annotation, *GrAPFI* ignore la même protéine si elle apparaît dans le voisinage. Par exemple, pour une protéine de requête q, *GrAPFI* collectera les voisins satisfaisant un score de similarité Jaccard maximum. Lorsque la similarité Jaccard maximale est inférieure à 1,0, *GrAPFI* omet les voisins avec exactement la même composition de domaine.

Le graphe d'apprentissage couvre 25 classes EC de niveau 2, 31 de niveau 3 et 408 de niveau 4 à partir de 41,618 oxydoréductases, 70,530 transférases, 100,027 hydrolases, 14,677 lyases, 25,551 isomérases et 29,735 ligases qui sont liées à l'aide de 10,866 signatures InterPro.

Dans le graphe d'apprentissage, il y a :

1. 4.3 % des protéines sont des protéines à domaine unique, c'est-à-dire des protéines n'ayant qu'un seul domaine dans leur composition de domaine,
2. 5.7 % des protéines ont plus d'un numéro EC associés à eux
3. Environ 15% des nœuds d'entraînement ont des annotations EC incomplètes, c'est-à-dire que les numéros EC attribués à ces protéines n'ont pas les quatre chiffres.

Pour valider *GrAPFI*, nous avons utilisé six protéomes de référence de Uniprot-KB/SwissProt comme ensemble de test. Les protéomes de référence sont les suivants : (1) Rattus norvegicus (UP000002494) contenant 1,953 protéines, (2) Mus musculus (UP000000589) contenant 3,682 protéines, (3) Saccharomyces cerevisiae (UP000002311) contenant 1,581 protéines, (4) Homo sapiens (UP000005640) contenant 3,843 protéines, et (5) Arabidopsis thaliana (UP000006548) contenant 5,352 protéines. (6) E. Coli (UP000000625) contenant 1,465 protéines. Pour chacun des protéomes de référence, nous avons collecté les domaines InterPro et les numéros EC de Uniprot-KB/Swissprot. Nous n'avons conservé que les protéines qui ont au moins un domaine InterPro et sont annotées avec un seul numéro EC.

2.3.2 Préparation de données pour la prédiction des termes GO

Nous avons utilisé un jeu de données de 1000 protéines appelé MetaGo [Zhang et al., 2018]. Nous construisons le graphe en utilisant les données d'entraînement de la 3ème édition de la compétition CAFA (CAFA3) [3]. CAFA3 est une compétition bien connue qui vise à annoter des séquences de protéines. Habituellement, CAFA3 fournit des séquences cibles et d'entraînement pour créer des modèles d'annotation. Dans cette étude, pour construire le réseau, nous avons utilisé des séquences d'entraînement CAFA3 et nous avons collecté des informations sur le domaine et la famille de ces protéines auprès d'UniprotKB. Ensuite, nous avons construit le graphe des protéines d'entraînement CAFA3. Ce graphe contient environ 65,000 nœuds sous forme de protéines et une moyenne de 16 termes GO de vérité terrain par protéine.

Pour préparer l'ensemble de test, nous avons utilisé des séquences de référence MetaGO [Zhang et al., 2018] et exécuté l'outil InterProScan pour identifier les domaines et les informations

3. `https://biofunctionprediction.org/cafa/`

sur la famille à partir de la séquence. En utilisant les domaines et les informations sur la famille des protéines de test, nous exécutons *GrAPFI-GO* et d'autres outils d'annotation sur toutes les protéines de test et récupérons les annotations prédites. Nous appliquons ensuite ou non la méthode d'élagage sémantique et nous comparons l'efficacité de l'annotation à l'aide de métriques standards.

2.3.3 Analyse des performances des annotations EC

Pour valider les performances d'annotation de *GrAPFI*, nous avons calculé la précision, la macro-précision, le macro-rappel et le score macro-F1 à différents niveaux de numéro EC. Pour chaque séquence de requête, nous avons sélectionné uniquement l'annotation la mieux classée. La méthode de validation que nous avons utilisée est similaire à la méthode de validation croisée. Pour chaque protéome, lors de l'annotation d'une protéine, nous avons supprimé cette protéine de l'ensemble d'apprentissage afin qu'une cartographie directe ne soit pas présente. La formule suivante (telle qu'utilisée dans [Li et al., 2018a]) a été utilisée pour calculer les scores d'évaluation :

$$accuracy(y, y') = \frac{1}{N}\sum_{i=0}^{N-1} 1(y_i = y'_i), \tag{2.2}$$

Ici, y et y' sont respectivement la liste des vérités terrain et des annotations prédites. La précision est calculée pour chaque niveau d'annotation EC. Comme il s'agit d'un problème de classification multi-classes, nous avons calculé le score de macro-précision, de macro-rappel et de macro-F1 comme suit :

$$Macro - precision(y, y') = \frac{1}{|M|}\sum_{l \in M} precision(y_l, y'_l), \tag{2.3}$$

$$Macro - recall(y, y') = \frac{1}{|M|}\sum_{l \in M} recall(y_l, y'_l), \tag{2.4}$$

$$Macro - F1(y, y') = \frac{1}{|M|}\sum_{l \in M} F1\ measure(y_l, y'_l), \tag{2.5}$$

Ici, y_l est la partie de y avec l'étiquette l et y'_l est la partie de y' avec l'étiquette l. M est

l'ensemble des classes. En général, la précision, le rappel et la mesure F1 sont calculés comme suit lorsque deux ensembles A et P sont donnés :

$$precision = \frac{|A \cap P|}{|P|}, \tag{2.6}$$

$$recall = \frac{|A \cup P|}{|A|}, \tag{2.7}$$

$$F1 - measure = \frac{2 \times precision \times recall}{precision + recall}. \tag{2.8}$$

Ici, A est l'ensemble des vérités de terrain et P est l'ensemble des prédictions. Comme les numéros EC sont hiérarchiques avec 4 niveaux, nous rapportons la précision, le rappel et la mesure F1 par niveau. Le niveau 1 désigne la classe principale, le niveau 2 la sous-classe, le niveau 3 la sous-sous-classe et le niveau 4 la sous-sous-sous-classe. Nous rapportons également la couverture qui est calculée selon $Coverage = M/T$, où T est le nombre total de protéines dans l'ensemble de test et M est le nombre de protéines pour lesquelles au moins un numéro EC est prédit. Pour chaque séquence de requête, nous considérons l'annotation la plus élevée. À des fins d'évaluation, nous avons divisé l'annotation EC à 4 chiffres en ses éléments constitutifs. Ensuite, pour le niveau 1, nous considérons le premier chiffre, pour le niveau 2, nous prenons les 2 premiers chiffres, pour le niveau 3, nous prenons les 3 premiers chiffres et enfin pour le niveau 4, nous prenons les quatre chiffres ensemble.

Pour l'ensemble de données de validation, *GrAPFI* a été exécuté en ajustant différents indices de similarité allant de 0,05 à 0,5, et en fixant une limite supérieure de similarité à 1 ou moins de 1.

La Figure 2.3 montre les performances de *GrAPFI* pour le protéome de référence d'A. thaliana.

Dans la Figure 2.3, nous montrons la précision des annotations (axe Y) par rapport à différents seuils de similarité Jaccard (axe X) pour les protéomes respectifs. Nous avons considéré des seuils de similarité allant de 0,05 à 0,5 car la couverture des annotations chute significativement après 0,5. Pour chacun des seuils, nous présentons la précision de la prédiction des chiffres EC-1, EC-2,

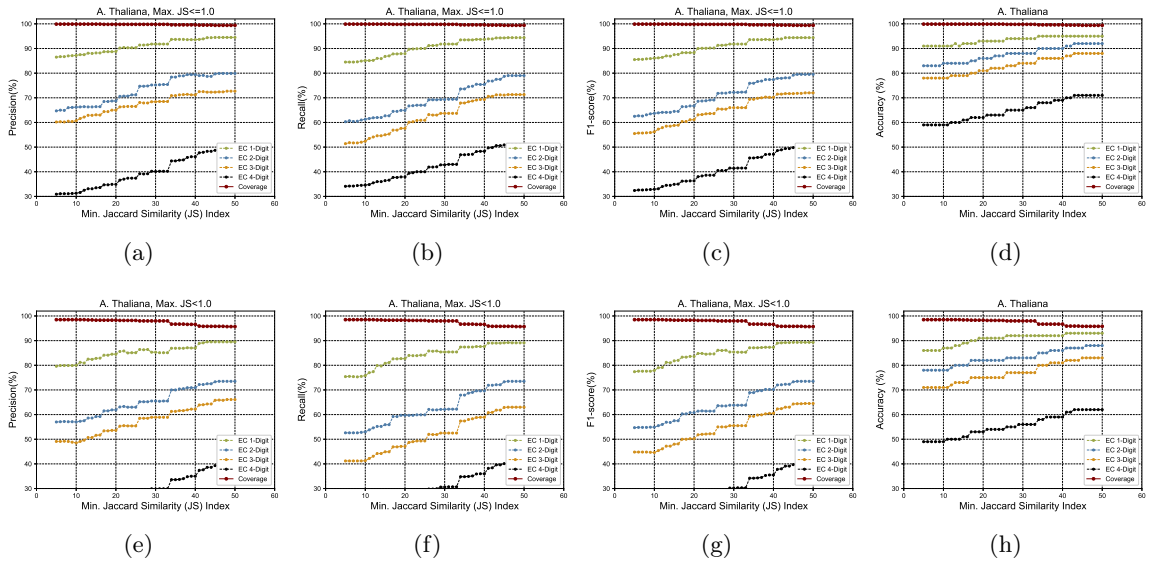
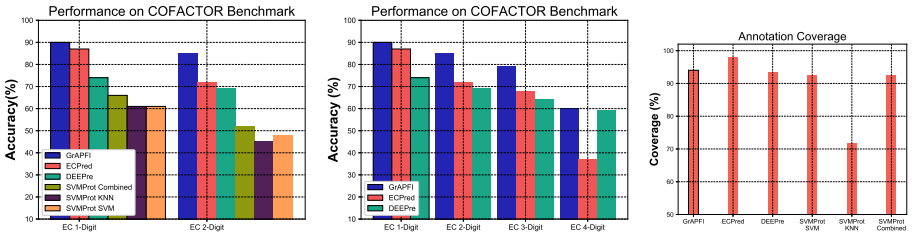


FIGURE 2.3 – Performances et couvertures en fonction de divers seuils des indices de similarité de Jaccard pour le protéome de référence *A. thaliana.* [Sarker et al., 2020b]

EC-3 et EC-4, respectivement en vert, bleu, orange et noir. Parallèlement à la précision, nous présentons également la couverture de l'annotation (courbe rouge). Pour chacune des figures, nous avons deux parties. La première partie montre la précision et la couverture en considérant uniquement les voisins qui ont une similarité Jaccard inférieure ou égale à 1. La deuxième partie considère la similarité Jaccard strictement inférieure à 1. On peut voir à partir de ces chiffres que *GrAPFI* fonctionne très bien pour tous les cas avec une bonne couverture.

Pour comparer *GrAPFI* avec d'autres méthodes, nous avons considéré trois méthodes basées sur l'apprentissage automatique, à savoir ECPred [Dalkiran et al., 2018], *DEEPre* [Li et al., 2018a] et *SVMProt* [Cai et al., 2003]. Les performances sont comparées sur la base du benchmark COFACTOR [Roy et al., 2012] comportant 318 séquences. Les résultats de la prédiction *SVMProt* couvrent trois conditions différentes : (1) en utilisant SVM uniquement, (2) en utilisant KNN uniquement et (3) en utilisant SVM, KNN et PNN combinés. La Figure 2.4 montre l'analyse des performances pour la prédiction de niveau EC-1 et de niveau EC-2. Les résultats présentés ici sont obtenus en utilisant une valeur d'indice de similarité Jaccard entre 0,3 et 1. Un seuil de similarité beaucoup plus bas engendre des faux positifs qui réduisent considérablement la précision. Sur la base des résultats obtenus, un indice de similarité de 0,3 permet d'obtenir un bon compromis entre précision et couverture.



(a) (b) (c)

FIGURE 2.4 – La partie 2.4(a) montre une comparaison des performances de *GrAPFI* avec *SVMProt* (SVM, KNN et mixte), *DEEPre* et *ECPred* pour les codes à 2 chiffres. Prédictions du nombre EC. La partie 2.4(b) montre la comparaison de précision de *GrAPFI* avec *DEEPre* et *ECPred* pour les 4 niveaux de prédiction EC. La partie 2.4(c) montre la couverture des méthodes considérées. [Sarker et al., 2020b]

Étant donné que toutes les méthodes ne peuvent pas faire de prédictions pour les quatre niveaux EC, nous avons comparé *GrAPFI* uniquement avec *ECPred* et *DEEPre* pour les nombres EC à 4 chiffres, comme le montre la Figure 2.4(b). Sur la Figure 2.4(c), nous montrons la couverture d'annotation des méthodes considérées ici. Elle montre qu'*ECPred* a une couverture supérieure par rapport aux autres méthodes. La raison pour laquelle *GrAPFI* ne parvient pas à atteindre la couverture la plus élevée est due au fait qu'il s'agit d'une méthode d'annotation basée sur le voisinage. *GrAPFI* effectue la propagation des étiquettes en filtrant les voisins faiblement liés et déterminés par un seuil de similarité minimum. En raison de cette action de filtrage, dans quelques cas, *GrAPFI* ne parvient pas à suggérer une annotation appropriée pour les protéines de requête. Cela réduit la couverture totale des annotations. Cependant, d'un autre côté, *GrAPFI* augmente la précision en considérant des voisins fortement liés. Comme le montrent les figures 2.4(b) et 2.4(c), *GrAPFI* a une meilleure précision par rapport à ECPred et *DEEPre*, mais il offre une couverture légèrement inférieure à ECPred.

Les performances de *GrAPFI* pour les protéomes de référence d'Homo sapiens, S. cerevisiae, Mus musculus, Rattus norvegicus et E. Coli ainsi qu'une évaluation des performances de *GrAPFI* pour la classification binaire (enzyme ou non) des protéines peuvent être consultées dans l'article [Sarker et al., 2020a].

2.3.4 Analyse des performances des annotations GO

Nous avons évalué l'effet de l'ajout des étapes d'élagage sémantique et de post-traitement sur les performances d'annotation de *GrAPFI-GO* et des autres outils d'annotation facilement disponibles, à savoir PANNZER et DeepGOPlus [Kulmanov and Hoehndorf, 2020].

TABLE 2.1 – Effet de la méthode d'élagage et de post-traitement sur les performances de *GrAPFI-GO* et de deux autres outils d'annotation automatique. La précision moyenne, le rappel et le score F1 sont calculés pour chaque méthode dans quatre situations. Pas de post-traitement : sans post-traitement ni élagage ; SS-max : élagage en utilisant le SS le plus élevé comme seuil ; SS-5 : élagage en utilisant le 5ème SS le plus élevé comme seuil ; SS-5-MS-max/2 : élagage en utilisant le 5ème SS le plus élevé et (MS maximum)/2 comme seuils. [Sarker et al., 2022]

Méthode	Post-traitement	Précision	Rappel	Mesure F1
GrAPFI	Sans post-traitement	0.165	0.108	0.107
	SS-max	**0.573**	0.115	0.175
	SS-5	0.445	0.380	0.376
	SS-5-MS-max/2	0.440	**0.391**	**0.379**
PANNZER	Sans post-traitement	0.547	**0.942**	0.668
	SS-max	**0.637**	0.225	0.301
	SS-5	0.634	0.515	0.536
	SS-5-MS-max/2	0.603	**0.689**	**0.609**
DeepGOPlus	Sans post-traitement	0.053	**0.653**	0.095
	SS-max	**0.249**	0.120	0.138
	SS-5	0.186	0.182	0.160
	SS-5-MS-max/2	0.167	0.233	**0.1725**

Nous exécutons les trois outils d'annotation sélectionnés sur les données de référence MetaGO et obtenons des ensembles de termes GO prédits. Ces ensembles prédits sont ensuite élagués à l'aide de la similarité sémantique et enrichis par un post-traitement hiérarchique. Les résultats sont présentés dans le tableau 2.1. Ici, nous présentons les métriques d'évaluation sans les catégoriser en aspects GO, car la méthode d'élagage et de post-traitement en plusieurs étapes prend énormément de temps pour produire l'ensemble d'annotations prédit. Pour chaque protéine de test, le score du modèle (MS, fourni par chaque méthode) et le score de similarité sémantique (SS, calculé avec notre implémentation de GOGO) sont obtenus pour tous les termes prédits pour cette protéine. Différents seuils de ces deux scores sont utilisés pour l'analyse. L'ensemble final de termes prédits est comparé à la vérité terrain du benchmark. Pour chaque outil d'annotation, le tableau 2.1 montre les performances d'annotation dans quatre cas : (1) sans aucun

type d'élagage ni de post-traitement, (2) lorsque le score SS le plus élevé est le seuil pour SS, (3) lorsque le 5ème score SS le plus élevé est le seuil pour SS et (4) lorsque le 5ème score SS le plus élevé et la moitié du score MS maximal sont les seuils pour SS et MS respectivement.

Les résultats expérimentaux montrent clairement que l'approche de post-traitement qui utilise la similarité sémantique des termes GO prédits pour élaguer l'ensemble prédit ainsi que l'enrichissement par les ancêtres, améliore considérablement les performances globales de la méthode *GrAPFI-GO*. En particulier, elle améliore la précision de plusieurs plis, passant de 16,5 % à 57,3 % lors de l'utilisation de SS-max comme coupure lors du post-traitement. De même, cette approche proposée améliore considérablement la précision de PANNZER et DeepGOPlus. Des expérimentations supplémentaires sont disponibles dans l'article [Sarker et al., 2022].

2.4 Discussion et conclusions

L'utilisation des graphes pour représenter les protéines est une approche intéressante, car la structure riche des graphes permet de relier non seulement les protéines qui interagissent, mais aussi celles qui partagent des informations similaires. La représentation des protéines avec des graphes permet aussi d'appliquer une panoplie d'algorithmes de graphes sur des données protéiques. Dans le contexte de ce travail, les algorithmes *GrAPFI* et *GrAPFI-GO* pour la propagation d'étiquettes ont été appliqués sur un graphe de protéines pour pouvoir annoter des protéines par des propriétés fonctionnelles décrivant leurs fonctions. Cette tâche d'annotation automatique des fonctions des protéines est un sujet important pour la communauté scientifique dans le domaine de la bioinformatique, en raison du manque d'annotation des protéines et en raison des coûts élevés et de la nature chronophage de l'annotation fonctionnelle manuelle.

Dans la Section 2.2.1, nous avons présenté la nouvelle façon de connecter les protéines sur la base de domaines potentiellement liés à leurs fonctions. Nous avons présenté dans la Section 2.3 les résultats expérimentaux qui motrent l'efficacité des méthodes proposées (*GrAPFI* et GrAPFI-GO) pour annoter des protéines avec des numéros EC ou par des termes GO. Cela signifie finalement que les méthodes d'annotation proposées sont biologiquement significatives. L'un des principaux avantages de l'utilisation de ces méthodes pour annoter les protéines est qu'ils produisent des annotations automatiques de qualité et explicables avec un pipeline d'an-

notation relativement à la portée de nos investigations et contributions. Bien que les méthodes d'annotation proposées dans le cadre de la thèse de Bishnu Sarker [Sarker, 2021] fonctionnent bien, leur utilisation présente quelques inconvénients. Par exemple, l'étape cruciale de construction du graphe de protéines se base sur la recherche de domaines des protéines qui doit être réalisée à l'aide d'un autre outil. En plus, ces méthodes ne peuvent pas être utilisées avec des protéines dépourvues d'informations sur le domaine et pour les protéines à domaine unique. Dans la plupart des cas, ces méthodes ne parviennent pas à trouver les annotations appropriées. Pour résoudre ce problème de dépendance de ces approches aux domaines protéiques, nous avons tenté, dans le cadre d'un travail de recherche, d'explorer les autres attributs tels que les voies biologiques, le génotype, le phénotype, le taxon qui pourraient incorporer des informations importantes dans le processus d'annotation. Plus particulièrement, nous avons construit un graphe de connaissances (KG) pour intégrer diverses sources de données et nous avons proposé une méthode préliminaire appelée Prot-A-GAN basée sur les réseaux antagonistes génératifs (GAN) [Goodfellow et al., 2014] pour l'annotation des protéines. En suivant les terminologies du GAN, nous formons d'abord un discriminateur en utilisant un échantillonnage négatif adaptatif au domaine pour discriminer les triplets positifs et négatifs, puis nous formons un générateur pour guider une marche aléatoire sur le graphe de connaissances qui identifie les chemins entre les protéines et les annotations GO [Sarker et al., 2021]. Bien qu'il s'agisse d'une méthode de preuve de concept, les résultats préliminaires de la méthode Prot-A-GAN sont prometteurs et ouvrent plusieurs pistes de recherche dans le domaine de la prédiction des fonctions de protéines en utilisant les graphes de connaissances et les techniques d'apprentissage profond [Sarker et al., 2021].

Chapitre 3

Prédiction de liens dans des graphes de connaissances complexes

3.1 Introduction

Dans sa forme la plus générale, un graphe est constitué d'un ensemble de nœuds représentant des objets ou des entités et d'un ensemble de liens représentant les relations entre les paires d'entités. Les graphes peuvent encoder de riches informations sur les nœuds et leurs relations complexes. Par exemple, le graphe d'amitié dans un réseau social, où les nœuds sont des personnes, les liens sont des amitiés entre des couples de personnes ; le réseaux d'interaction protéine/protéine (PPI), où les nœuds sont des protéines, et les liens sont des interactions entre des paires de protéines. Les graphes peuvent contenir des informations plus descriptives sur les nœuds, comme les types, les attributs et aussi sur les liens, comme les noms des liens. Ces types de graphes sont généralement connus sous le nom de graphes de connaissances (KG). Les directions et les noms sont deux exigences obligatoires des liens dans les graphes de connaissances. Ces graphes supportent des liens multiples entre une paire de nœuds ainsi que des boucles. Par conséquent, les graphes de connaissances encodent plus d'informations sémantiques que les graphes simples.

La plupart des graphes du monde réel sont de grande taille. Par exemple, le graphe HI-III PPI [Kovacs et al., 2019] est constitué de 18,000 protéines humaines et de 35,500 interactions protéine-protéine ; le graphe de connaissances Freebase [Google, 2013] contient près

de 44 millions de nœuds représentant des entités de divers domaines (par exemple, des personnes, des lieux, des religions, des entreprises) et 2,4 milliards de liens représentant des relations nommées entre des paires d'entités. Cependant, ces graphes sont loin d'être complets. Le lieu de naissance et la nationalité sont manquants pour environ 78,5 % et 93,8 % des personnes, respectivement, dans le graphe de connaissances Freebase [Min et al., 2013]. Le problème de la prédiction des liens manquants ou de l'inférence de nouveaux liens dans un graphe est communément appelé le problème de prédiction de liens [Lü and Zhou, 2011]. Formellement, la prédiction de liens est la tâche de prédire la probabilité d'un lien entre deux nœuds en se basant sur les informations topologiques et/ou structurelles disponibles d'un graphe [Xu et al., 2016]. Les approches de prédiction de liens permettent de découvrir des relations complexes dans un graphe, ce qui nous aide à prendre de meilleures décisions. Par exemple, la tâche de reprogrammation/repositionnement de médicaments peut être formulée comme une tâche de prédiction de liens dans un graphe biologique où les liens non observés entre les nœuds de médicaments (approuvés/essayés cliniquement) et de maladies sont prédits [Somolinos et al., 2021, Fiscon et al., 2021, Kanatsoulis and Sidiropoulos, 2021]. Les experts peuvent ensuite procéder à la validation expérimentale de ces paires médicament-maladie prédites afin de trouver de nouveaux médicaments efficaces et/ou appropriés pour la maladie en question.

Le développement d'une approche de prédiction de liens pour les graphes soulève deux défis majeurs. Le premier défi est de fournir des performances de prédiction élevées sur des graphes provenant de différents domaines. Le second défi est de fournir une explication suffisante des résultats pour apporter de la fiabilité à l'approche.

La prédiction de liens dans les graphes de connaissances est plus difficile que dans les graphes simples, car les graphes de connaissances contiennent des relations complexes entre les nœuds. Les approches de prédiction de liens pour les graphes de connaissances sont soit basées sur des règles logiques, soit basées sur les plongements. Des études montrent que les approches de prédiction de liens basées sur les plongements offrent de meilleures performances de prédiction et une meilleure évolutivité que les approches traditionnelles basées sur des règles. Cependant, les approches basées sur les plongements ne sont pas explicables. En fait, les approches de prédiction de liens basées sur les plongements dans les graphes de connaissances soulèvent deux nouveaux défis en

plus de la performance de prédiction élevée et de l'explicabilité. Le premier est la génération d'exemples négatifs de haute qualité, car ils ne sont pas facilement disponibles. Le second est le passage à l'échelle des approches de prédiction de liens, car les graphes de connaissances représentant le monde réel sont de taille énorme. Bien qu'il existe de nombreuses approches pour la prédiction de liens dans les graphes, il est remarquable que les approches existantes ne sont pas assez bonnes pour offrir des solutions satisfaisantes. Dans ce contexte, l'état de l'art de la prédiction de liens dans les graphes de connaissances appelle deux orientations de recherche : (1) la génération d'exemples négatifs de haute qualité pour les approches basées sur les plongements, (2) l'explication de la prédiction de liens basée sur les plongements.

Dans ce chapitre, je présente tout d'abord notre contribution pour la génération d'exemples négatifs de bonne qualité avec nos approches basées sur les plongements. Ensuite, je présente notre méthode générique à base de règles pour l'explication des résultats des méthodes de prédiction de liens dans les graphes de connaissances. À la fin de ce chapitre, je présente notre application des méthodes proposées pour le repositionnement de médicaments pour le COVID-19.

3.2 Échantillonnage négatif simple (SNS)

Au cours de l'apprentissage de plongement (ou *embedding*) efficace de graphe de connaissances, l'échantillonnage de triplets négatifs a été mis en évidence comme une étape importante, car les graphes de connaissances ne contiennent que des triplets positifs. Dans le cadre de la thèse de Kamrul Islam [Islam, 2022], nous avons proposé une nouvelle méthode efficace d'échantillonnage négatif simple (SNS pour *Simple Negative Sampling*) pour échantillonner des triplets négatifs de haute qualité dans les graphes de connaissances.

Dans cette section, nous décrivons d'abord un modèle de plongement géométrique de graphe de connaissances, puis notre méthode SNS. Dans ce qui suit, nous utilisons $\mathbb{E}$ pour désigner l'ensemble de toutes les entités, $\mathbb{R}$ pour désigner l'ensemble de toutes les relations, $\mathbb{S}$ pour désigner l'ensemble de triplets positifs d'entrainement, $\mathbb{D}$ pour désigner l'ensemble de triplets de test positifs, $\mathbb{Q}$ pour désigner l'ensemble de tous les triplets positifs, d pour désigner la taille de la dimension du plongement, m pour désigner la taille du lot, S_m et S'_m pour désigner un lot de

triplets positifs et négatifs respectifs.

Le modèle de KGE géométrique commence par initialiser les plongements d'entités et de relations de manière aléatoire à partir de distributions uniformes/gaussiennes [Wang et al., 2017]. Pour l'apprentissage du modèle, un lot de triplets d'entraînement positifs S_m est extrait de l'ensemble de triplets d'entraînement, $\mathbb{S}$. Une méthode d'échantillonnage négatif est ensuite utilisée pour générer un lot de triplets négatifs S'_m pour le lot de triplets positifs S_m. Les lots de triplets positifs et négatifs sont ensuite utilisés par la méthode d'apprentissage par paires (ou *pairwise learning*) [Menon and Elkan, 2011] pour apprendre les plongements. Dans l'apprentissage par paires, le modèle tente d'attribuer un score de plausibilité plus élevé à un triplet positif qu'à son triplet négatif correspondant. La perte par paires (ou *pairwise loss*) pour un triplet positif $(h, r, t) \in S_m$ et son triplet négatif correspondant $(h', r, t') \in S'_m$ est définie dans l'équation 3.1 [Wang et al., 2017].

$$L(f(h, r, t), f(h', r, t')) = \Big[\lambda - f(h, r, t) + f(h', r, t')\Big]_+ \tag{3.1}$$

L est la fonction de perte, f est la fonction de score du modèle de plongement, λ est la marge et $[.]_+ = max(0, .)$ est la fonction de perte (ou *hinge function*). L'objectif de l'apprentissage est d'optimiser les plongements d'entités et de relations pour minimiser la perte totale par paires par l'équation 3.2.

$$\min_{\Theta} \sum_{\forall_{(h,r,t)\in S_m, (h',r,t')\in S'_m}} L(f(h, r, t), f(h', r, t')) + \lambda reg(\Theta) \tag{3.2}$$

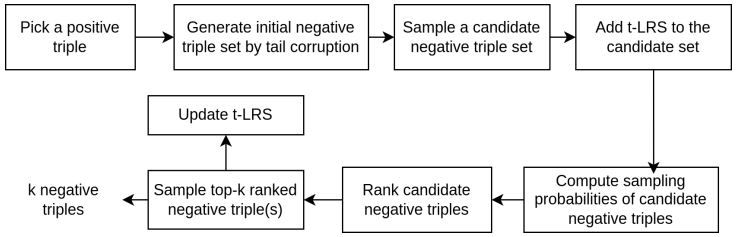
L est la fonction de perte, f est la fonction de score du modèle de plongement, λ est la marge, $reg(\Theta)$ est le terme de régularisation, $(h, r, t) \in S_m$ est un triplet positif et $(h', r, t') \in S'_m$ est le triplet négatif correspondant.

Le processus de mise à jour des plongements est répété pour tous les lots de triplets positifs, et l'ensemble du processus d'apprentissage est répété pendant T fois/époques (ou *epochs*). Le processus d'entraînement du modèle est similaire à un entraînement du modèle de plongement géométrique traditionnel, sauf que nous adaptons notre méthode d'échantillonnage négatif.

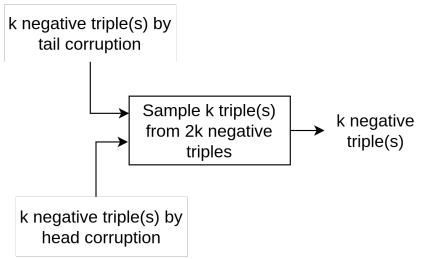
La méthode SNS vise à générer des triplets négatifs de haute qualité tout en évitant le problème de «vanishing-gradient» de l'échantillonnage aléatoire uniforme, le problème complexe d'optimisation des paramètres de l'échantillonnage basé sur les GANs et les problèmes de consommation excessive de mémoire de NSCaching. La Figure 3.1(a) montre les étapes éssentielles de l'échantillonnage avec SNS. Les étapes pour chaque triplet positif sélectionné $q = (h, r, t) \in \mathbb{S}$ sont décrites ci-dessous.

Étape 1. Génération initiale d'un ensemble de triplets négatifs : Comme illustré dans la Figure 3.1(a), SNS commence par générer un ensemble initial de triplets négatifs pour le triplet positif (q) par perturbation. Cette étape est similaire aux autres méthodes d'échantillonnage. Avec la méthode de perturbation, la tête (h)/queue (t) du triplet positif est corrompue en remplaçant la tête/queue par d'autres entités de l'ensemble d'entités ($\mathbb{E}$). En même temps, on vérifie que l'ensemble négatif ne contient aucun triplet positif.

Pour des raisons de simplification, nous considérons uniquement la perturbation de la queue. Par conséquent, corrompre la queue (t) donne l'ensemble négatif initial $q'_0(t) = \{(h, r, t') \notin \mathbb{Q} | t' \in \mathbb{E}\}$.



(a) Génération de k triplets négatifs de haute qualité en corrompant la queue (k triplets négatifs sont également générés en corrompant la tête



(b) Échantillonage de k triplets négatifs à partir de 2k négatifs de bonne qualité

FIGURE 3.1 – La méthode d'échantillonnage SNS. [Islam et al., 2022]

Étape 2. Génération de l'ensemble de triplets candidats : Généralement, la taille de l'ensemble $q'_0(t)$ est grande car le graphe de connaissances contient un grand nombre d'entités. Il a été démontré que seuls certains des négatifs initiaux de l'ensemble sont de bonne qualité [Zhang et al., 2019]. Comme nous avons besoin de peu de négatifs pour chaque triplet positif, nous échantillonnons aléatoirement N_1 triplets à partir de $q'_0(t)$ pour générer l'ensemble de triplets négatifs candidat $q'_1(t)$(*c'est-à-dire*, $q'_1(t) \subseteq q'_0(t)$) pour le paramètre défini par l'utilisateur N_1.

Au fur et à mesure de l'apprentissage, on souhaite que la qualité des triplets négatifs pour l'étape suivante soit meilleure ou proche de la qualité du négatif ou des négatifs de l'étape en cours. À cette fin, les triplets négatifs échantillonnés pour chaque triplet positif de l'époque actuelle sont stockés dans une structure de données appelée LRS (pour *Least Recently Selected*). Supposons que $LRS_q = (h - LRS_q) \cup (t - LRS_q)$ stocke les triplets négatifs pour le triplet positif (q) où $h - LRS_q$ et $t - LRS_q$ sont les triplets négatifs échantillonnés par perturbation de la tête et de la queue respectivement à l'époque précédente. Les triplets négatifs enregistrés sont utilisés dans l'étape en cours pour échantillonner des triplets négatifs de haute qualité. Comme décrit par le processus de génération de triplets négatifs par perturbation de la queue, $t - LRS_q$ est ajouté à l'ensemble de triplets négatifs candidat, $q'_1(t)$, *c'est-à-dire*, $q'_1(t) := q'_1(t) \cup t - LRS_q$. Le but de l'utilisation de LRS est de favoriser le comportement d'exploration de la méthode SNS et de garantir que la qualité des triplets négatifs à l'étape en cours est meilleure ou au moins proche de la qualité des triplets négatifs à l'étape précédente.

Étape 3. Calcul de la probabilité d'échantillonnage :

Dans cette étape, nous calculons la probabilité d'échantillonnage de chaque triplet négatif dans l'ensemble de triplets négatifs candidats $q'_1(t)$. Les triplets négatifs avec des probabilités plus élevées sont considérés comme des triplets négatifs de haute qualité. La probabilité est définie en fonction du score de distance de chaque triplet négatif. Le score de distance entre un triplet négatif $(h, r, t'_i) \in q'_1(t)$ et le triplet positif associé $(q = (h, r, t))$, est calculé comme la distance

entre l'entité corrompue (t) et la nouvelle entité (t') (voir formule 3.3).

$$d(\ (h,r,t),(h,r,t'_i)\ ) = ||\mathbf{t} - \mathbf{t'_i}|| \tag{3.3}$$

où, $\mathbf{t}$ et $\mathbf{t'_i}$ sont les plongements des entités t et t'_i. Une fonction softmax est ensuite utilisée pour calculer le score de probabilité d'échantillonnage de chaque triplet négatif candidat $(h,r,t'_i) \in q'_1(t)$ comme illustré par la formule 3.4.

$$P(h,r,t'_i) = \frac{exp(\frac{1}{d(t,t'_i)})}{\sum_{j=1}^{N_1} exp(\frac{1}{d(t,t'_j)})} \tag{3.4}$$

La fonction *softmax* calcule une probabilité d'échantillonnage plus élevée pour les triplets négatifs candidats avec un score de distance faible pour étayer notre hypothèse.

Étape 4. Échantillonnage de triplets négatifs et mise à jour de LRS :

Les triplets de l'ensemble des candidats négatifs, $q'_1(t)$ sont classés par ordre décroissant de leurs probabilités et k triplets négatifs sont échantillonnés. Un choix naturel pourrait être d'échantillonner top-k (k=1 pour l'apprentissage par paires, $k > 1$ pour l'apprentissage par maximum de vraisemblance) triplets négatifs. Cependant, l'échantillonnage des triplets négatifs classés parmi les k premiers pourrait poser deux problèmes. Premièrement, il y a une chance d'échantillonnage répété du ou des mêmes triplets négatifs (même dans des étapes consécutives) car l'ensemble de triplets négatifs candidats actuel comprend également le ou les triplets négatifs de haute qualité échantillonnés récemment (LRS). Cela affecterait la capacité d'exploration de l'échantillonnage de la méthode SNS. Deuxièmement, l'existence de faux triplets négatifs n'est pas ignorable car les triplets négatifs et positifs ont des scores élevés [Wang et al., 2017]. Pour minimiser ces problèmes, SNS échantillonne au hasard k triplets négatifs à partir de N_2 triplets les mieux classés dans $q'_1(t)$ comme $q'_t = \{(h,r,t'_i)|rank_{(h,r,t'_i)} \leq N_2\}$ pour un paramètre défini par l'utilisateur N_2 où $N_2 > k$. Comme SNS échantillonne des triplets négatifs à partir de N_2 triplets négatifs les mieux classés, SNS évite les faux triplets négatifs les mieux classés pour de nombreux triplets positifs.

SNS répète les étapes décrites ci-dessus (de la génération initiale de l'ensemble de triplets né-

gatifs à l'échantillonnage de k triplets négatifs) pour la corruption de tête(h) également pour échantillonner k triplets négatifs comme q'_h pour le triplet positif, q. Le LRS_q est mis à jour avec les triplets négatifs échantillonnés de haute qualité 2k comme $h - LRS_q = q'_h,\ t - LRS_q = q'_t$. Enfin, nous échantillonnons au hasard k triplets négatifs de haute qualité comme $q'_{h,r,t}$ à partir de 2k triplets négatifs dans $q'_h \cup q'_t$ (Figure 3.1(b)). Le ou les k triplets négatifs échantillonnés, q' et le triplet positif correspondant, q sont ensuite ajoutés à l'ensemble d'apprentissage pour les modèles de plongement de graphes de connaissances.

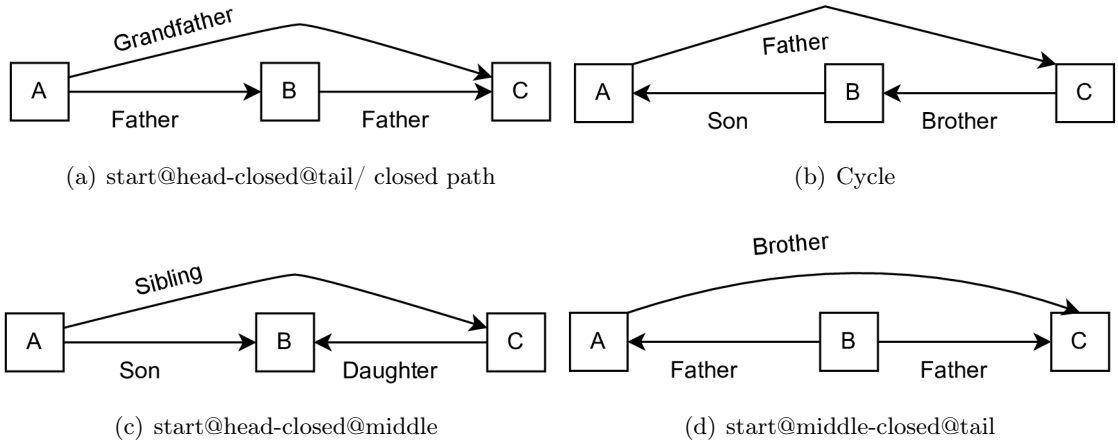
Il est important de mentionner que la méthode SNS est générale et injectable à n'importe quelle méthode de plongement géométrique de graphes de connaissances pour échantillonner les triplets négatifs.

3.3 Explication des prédictions

L'applicabilité des approches de prédiction de liens dans des graphes de connaissances est souvent limitée en raison de leur incapacité à expliquer les prédictions. Nous avons proposé dans le cadre de la thèse de Kamrul Islam [Islam, 2022], une nouvelle méthode pour apprendre des règles de haute qualité basées sur un graphe de connaissances et ses plongements appris. Ensuite, nous utilisons une stratégie abductive pour expliquer les prédictions faites par la méthode des plongements basée sur l'instanciation des règles extraites pour trouver des chemins explicatifs pour les prédictions.

Pour extraire des règles à partir d'un graphe de connaissances, nous considérons différents modèles de chemins existants dans le graphe. La Figure 3.2 illustre quatre types de chemins de longueur 2 dans le graphe 'Family', un graphe de connaissances, où les relations sont les relations familiales entre les personnes [Kok and Domingos, 2007, Sadeghian et al., 2019]. Les méthodes existantes d'extraction de règles basées sur les plongements ne gèrent que le premier type, *c'est-à-dire*, chemin fermé (Figure 3.2(a)) et les trois autres types (Figures 3.2(b),3.2(c),3.2(d)) ne sont pas gérés.

Nous nous concentrons sur les règles qui consistent en une relation de tête H et une séquence



(a) start@head-closed@tail/ closed path

(b) Cycle

(c) start@head-closed@middle

(d) start@middle-closed@tail

FIGURE 3.2 – Différents types de chemins pour définir des règles de longueur 2. [Islam et al., 2022]

de relations de corps $B_1, B_2, ..., B_n$ sous la forme suivante :

$$B_1(e_0, e_1) \wedge B_2(e_1, e_2) \wedge ..... \wedge B_n(e_{n-1}, e_n) \rightarrow H(e_0, e_n)$$

où e_i est une variable d'entité, H est la relation tête/cible de e_0 à e_n et $B_1, B_2, ..., B_n$ sont la séquence des relations de corps. Dans le corps de la règle, $B_i(e_{i-1}, e_i)$ est soit une relation (*c'est-à-dire*, (e_{i-1}, r_i, e_i)) soit son inverse (*c'est-à-dire*, (e_{i-1}, r_i^{-1}, e_i)).

Un chemin simple est une séquence de triplets dans laquelle une entité commune apparaît entre deux triplets consécutifs sans cycle. Nous accordons plus d'importance aux chemins courts qu'aux longs. Nous calculons le score d'un chemin P comme le score moyen sur tous les triplets composant le chemin, pénalisé par la longueur du chemin. Nous définissons la fonction de score de chemin PS comme défini dans l'équation 3.5.

$$PS(P) = \frac{1}{l^{1+\alpha}} \sum_{i=1}^{l} f(h_i, r_i, t_i) \tag{3.5}$$

où f est la fonction de score de plongement, l est la longueur du chemin, α est le facteur pénalisant et $0 \leq \alpha \leq 1$. $\alpha = 0$ signifie que nous considérons tous les chemins de la même manière quelle que soit leur longueur alors que des valeurs plus grandes donnent plus d'importance aux chemins plus courts. Nous supposons qu'un chemin avec un score élevé est un bon chemin par rapport au plongement en question.

Dans ce qui suit, nous décrivons la procédure d'extraction de règles qui commence par la génération de graphes et se poursuit par l'échantillonnage de triplets et la fouille de règles.

3.3.1 Génération d'un graphe bidirectionnel

Pour faciliter la recherche des chemins à utiliser dans les corps de règles, nous générons un graphe bidirectionnel G à partir des triplets appris à partir du graphe de connaissances. Pour chaque triplet $(h, r, t) \in G$, on ajoute un triplet avec une relation inversée, *c'est-à-dire*, $G \leftarrow G \cup (t, r^{-1}, h)$ s'il n'y a pas de relation de t à h. De cette façon, nous transformons le graphe bidirectionnel de sorte que pour chaque triplet, la queue soit accessible depuis la tête ainsi que la tête soit accessible depuis la queue. Cette transformation nous aide à extraire divers types de chemins.

3.3.2 Fouille de règles

Nous schématisons la procédure de fouille de règles pour une relation spécifique r dans la Figure 3.3. Dans ce qui suit, nous décrivons les différentes étapes de la procédure.

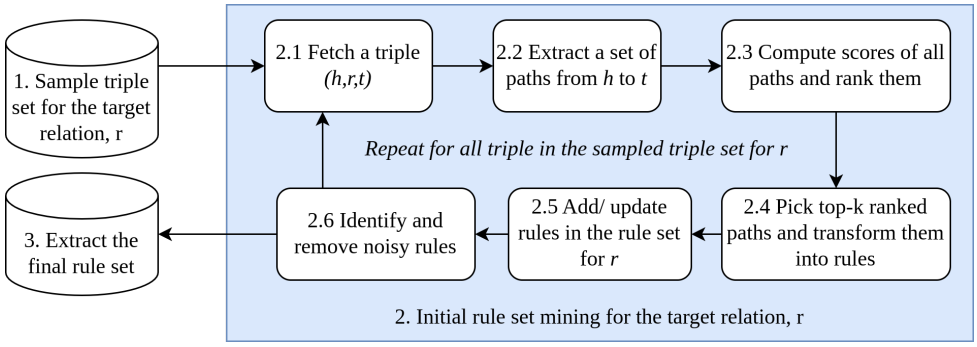


FIGURE 3.3 – Fouille de règles pour une relation cible r. [Islam et al., 2022]

Étape 1 : Échantillonnage de triplets

Pour extraire des règles, nous sélectionnons aléatoirement un sous-ensemble de triplets de la relation cible r.

Étape 2 : Fouille d'un ensemble de règle initial

La procédure initiale de fouille des règles pour une relation r est illustrée dans la Figure 3.3 (en bleu).

— **Étape 2.1 : Récupération d'un triplet**

La procédure commence par la récupération du triplet arrivé (h, r, t). Notez qu'un seul triplet est extrait à la fois et que le triplet suivant est extrait après le traitement complet du triplet actuel.

— **Étape 2.2 : Extraction de chemins**

Après avoir récupéré le triplet, les relations r de h à t et son inverse(r^{-1}) (si elle existe) sont temporairement supprimées du graphe. Ensuite, un ensemble de chemins simples jusqu'à une longueur, MaxL de h à t sont extraits. Ici, MaxL est la longueur de corps maximale prédéfinie, ou simplement la longueur de règle maximale. Après avoir extrait l'ensemble de chemins, les relations supprimées entre h et t sont à nouveau ajoutées au graphe.

— **Étape 2.3 : Calcul des scores et des classements des chemins**

Les scores de tous les chemins extraits sont calculés à l'aide de la fonction de score des chemins dans l'équation 3.5. Notez qu'un chemin peut contenir la relation inverse de la forme (h_i, r_i^{-1}, t_i). Dans ce cas, le score du triplet est calculé comme $f(t_i, r_i, h_i)$. Ce calcul nous permet d'inclure tous les cas de la Figure 3.2.

Les chemins sont ensuite classés par ordre décroissant de leur score. Un meilleur classement d'un chemin signifie une meilleure chance que le chemin génère une bonne règle.

— **Étape 2.4 : Définition des règles à partir des chemins**

Nous définissons les chemins les mieux classés comme des supports solides pour le triplet actuel où k est un paramètre défini par l'utilisateur. Dans cette étape, un chemin est transformé en règle en remplaçant les entités de triplets par des variables. Si la tête et la queue d'un triplet (h_i, r_i, t_i) sont remplacées par les variables e_{i-1} et e_i alors le triplet est transformé en la relation de corps sous la forme $r_i(e_{i-1}, e_i)$. Une relation inverse (h_i, r_i^{-1}, t_i) est transformée sous la forme $r_i(e_i, e_{i-1})$.

— **Étape 2.5 : Mise à jour de l'ensemble de règles**

Si une règle de haute qualité, *Rule*, n'existe pas dans l'ensemble initial de règles, *Rule_ set* d'une relation, alors elle est ajoutée en initialisant sa structure comme suit :

$$Rule.count \leftarrow 1$$

$$Rule.energy \leftarrow 1.0 + \gamma$$

La variable 'count' de la règle représente le nombre d'occurrences de la règle et la variable

d'énergie prend l'énergie complète et un taux de décroissance supplémentaire (γ) comme énergie initiale. La règle est ensuite ajoutée au jeu de règles initial, *Rule_set c'est-à-dire*, $Rule\_set \leftarrow Rule\_set \cup Rule$. Dans le cas où la règle *Rule* existe déjà dans le *Rule_set*, alors son ordre ainsi que sa valeur d'énergie sont mises à jour comme suit :

$$Rule\_set[Rule].count \leftarrow Rule\_set[Rule].count + 1$$
$$Rule\_set[Rule].energy \leftarrow 1.0 + \gamma$$

— **Étape 2.6 : Identification et suppression des règles bruyantes**

Les règles qui ne sont pas utiles pour extraire des règles de haute qualité pour une relation sont identifiées comme bruyantes et supprimées immédiatement. Cette suppression réduit la consommation inutile de mémoire et de temps de calcul. Dans notre méthode, une règle relève de l'une des trois lois.

— Une règle 'potentielle' a une énergie positive et ordre égal à 1,

— une règle "stable" a un ordre supérieur à 1, et

— une règle bruitée a un ordre égal à 1 et une énergie négative.

Une règle 'potentielle' peut devenir 'stable' ou 'bruyante' plus tard. A chaque fois qu'un triplet est extrait de l'ensemble de triplets échantillonné, l'énergie de chaque règle potentielle, *Rule* du *Rule_set* est diminuée du taux de décroissance γ comme suit :

$$Rule\_set[Rule].energy \leftarrow Rule\_set[Rule].energy - \gamma$$

Habituellement, le taux de décroissance est $0 < \gamma < 1$. Une petite valeur de γ signifie que les règles potentielles seront vivantes plus longtemps avant de mourir si elles ne durent pas longtemps. Généralement, le nombre de chemins entre deux entités est élevé dans un graphe de connaissances dense et faible dans un graphe éparse. Par conséquent, nous soutenons que la possibilité de présence de règles bruitées pour une relation est élevée dans un graphe dense et faible dans un graphe éparse. Toutes les règles potentielles avec des énergies négatives sont identifiées comme des règles bruyantes et immédiatement supprimées de la mémoire.

Étape 3 : Extraction finale de l'ensemble de règles

Les processus de fouille de règles ci-dessus (étapes 2.1 à 2.6) sont répétés jusqu'à ce que tous les triplets de l'ensemble de triplets échantillonné soient traités. En fin de traitement, le *Rule_ set* contient des règles 'stables' et quelques règles 'potentielles'. Le jeu de règles final ne contient que les règles "stables".

L'algorithme 2 décrit les étapes d'extraction de l'ensemble de chemins classés et l'algorithme 3 décrit les étapes de fouille de l'ensemble de règles en fonction de l'ensemble de chemins extraits.

Algorithm 2 Extract_paths : Extracting paths for a given triple

Input: Bidirectional Knowledge graph(G), a triple (h,r,t), maximum path length (MaxL)
Output: Paths, Ranks

G←G-{h,r,t} `// temporarily remove the relation from h to t`
Path_Scores← ∅ Ranks← ∅ Paths=all_simple_paths(G, source=h, target=t, cutoff=MaxL) [4]
`/* extract all simple paths from h to t with maximum path length, MaxL */`
foreach *P∈Paths* **do**
    Score_P ← Score of the path, P `// using equation 3.5`
    Path_Scores ← Path_Scores ∪ Score_P
Ranks ← Ranks of paths based on Path_Scores `// in decreasing order of path scores`
G←G∪{h,r,t}

Algorithm 3 Learn_rule : Learning rules for a relation

Input: Target relation (r), Bidirectional Knowledge graph (G), Triple set (T_r) with r, Decay rate (γ) , Sampling rate (β), Top-k, maximum path length (MaxL)
Output: *Rule_ set*

T'_r ← Randomly sample ($|T_r| * \beta$) triples from T_r **foreach** *(h,r,t)*$\in T'_r$ **do**
    Paths, Ranks =Extract_paths(G,(h,r,t), MaxL) `// Call Algorithm 2 to extract paths`
    `/* If a path exists then update its info, otherwise add the path to PathMetaInfo if it is ranked in top-k */`
    foreach *P∈Paths* **do**
        Rule=Transform_path_to_rule(P) `// replace entities with literals`
        if *Rule*$\in$ *Rule_ set* **then**
            Rule_ set[Rule].Count← *Rule_ set*[Rule].Count + 1 `// increment counter`
            Rule_ set[Rule].Energy← $1 + \gamma$ `// reset energy to full and additional` γ
        else
            `/* add a new rule */`
            if *Ranks[Rule]*$\leq$*top-k* **then**
                Rule.Count← 1 Rule.Energy← $1 + \gamma$ *Rule_ set*[Rule]← *Rule_ set* ∪ Rule
    `/* Identify the noisy rules */`
    foreach *Rule*$\in$*Rule_ set* **do**
        Rule.Energy←Rule.Energy - γ **if** *Rule.Energy* ≤ 0 & *Rule.Count*≤ 1 **then**
            Rule_ set=*Rule_ set* - Rule `// remove the noisy rule`
`/* generate the final rule set by removing potential rules */`
foreach *Rule*$\in$*Rule_ set* **do**
    if *Rule.Count*≤ 1 **then**
        Rule_ set=*Rule_ set* - Rule `// remove the potential rule`

3.3.3 Abduction pour la prédiction de lien explicable

Une fois les règles générées, elles peuvent être exploitées pour fournir des explications plausibles à l'appui de certaines prédictions. Nous proposons une procédure inspirée du raisonnement abductif sur les règles pour déduire la ou les meilleures explications d'une prédiction. En effet, le raisonnement abductif utilisant une règle $body \rightarrow head$ nous permet de déduire (ou au moins de supposer) que le corps est satisfait à chaque fois que la tête est observée. Donc étant donné une prédiction comme une observation, on cherche des règles dont la tête correspond à la prédiction. Chaque instanciation de règle (utilisant les chemins de graphe bidirectionnels) constitue alors une explication plausible de la prédiction. Ainsi, nous avons un moyen de classer davantage les prédictions top-k en fonction de leur explicabilité [Islam et al., 2022].

3.4 Repositionnement de médicaments explicable avec évaluation au niveau moléculaire pour COVID-19

Dans un contexte de graphe de connaissances, le repositionnement des médicaments est formulé comme une tâche de prédiction de lien où la probabilité d'une relation *Treat* est calculée à partir d'un composé approuvé (tête) vers une maladie (queue) c'est-à-dire, calculer la probabilité d'un triplet *(Composé,Treat,Maladie)*. Sur la base de cette formulation, il existe plusieurs approches de repositionnement de médicaments dans la littérature. Peu d'entre elles sont génériques [Mohamed et al., 2020, Zhu et al., 2022a] et les autres sont spécifiques à certaines maladies [Gao et al., 2022, Sosa et al., 2019, Zhu et al., 2020, Ioannidis et al., 2020].

La pandémie de COVID-19, causée par le syndrome respiratoire aigu sévère (SARS-CoV-2), a coûté la vie à près de 6 millions de personnes en 2020 et 2021, et elle continue toujours. À la connaissance de la communauté scientifique et médicale, aucun médicament spécifique n'est disponible à ce jour contre le COVID-19. Il existe un certain nombre d'études de repositionnement de médicaments pour le COVID-19 [Ioannidis et al., 2020, Zhang et al., 2021, Yan et al., 2021a, Kanatsoulis and Sidiropoulos, 2021, Choudhary et al., 2021].

La plupart des approches utilisent des méthodes traditionnelles de plongement de graphes de connaissances pour apprendre les plongements d'entités et de relations dans des graphes

de connaissances biologiques centrés sur le COVID-19, comme le graphe "Drug Repurposing Knowledge Graph" (DRKG) [Ioannidis et al., 2020], puis utilisent les plongements dans la tâche de repositionnement de médicaments. L'évaluation des prédictions par rapport aux médicaments en cours d'essai pour le COVID-19 est le seul moyen d'évaluer l'efficacité de ces approches.

Dans ce travail, nous proposons une étude pour le repositionnement des médicaments pour la maladie COVID-19 tout en expliquant les résultats de l'étude. L'architecture de la méthode proposée est illustrée dans la Figure 3.4 [Islam et al., 2023]. Nous commençons par collecter et nettoyer le graphe DRKG. Ensuite, nous proposons une nouvelle approche pour générer un ensemble de plongements de haute qualité du graphe DRKG en utilisant trois méthodes de plongement traditionnelles et la méthode d'analyse en composantes principales (ACP). Les plongements sont utilisées pour former un modèle de prédiction basé sur un réseau neuronal profond (DNN). Le modèle formé est utilisé pour prédire la probabilité de tous les triplets *(Composé, Treat, COVID-19)* non observés où COVID-19 est représenté par 27 protéines associées au SARS-CoV-2. Les triplets sont ensuite classés par ordre décroissant de leurs valeurs de probabilité et les 100 meilleurs composés sont prédits comme composés potentiels pour le COVID-19. Les 100 meilleures prédictions sont évaluées sur la base de deux groupes de méthodes : (1) l'appariement croisé avec des médicaments en cours d'essai pour le COVID-19 et (2) l'évaluation moléculaire basée sur les structures des composés et des protéines. Outre ces évaluations, nous apprenons des règles de haute qualité au sein du DRKG et fournissons des explications possibles pour les prédictions sélectionnées basées sur nos méthodes de fouille et d'explication de règles décrites dans la Section 3.3).

Cette étude se distingue des approches de l'état de l'art sur trois points majeurs. Premièrement, les études existantes dépendent d'un seul modèle de plongement de graphes de connaissances ignorant le fait qu'un seul modèle ne peut apprendre qu'un bon plongement de certains types de relations [Rossi et al., 2021, Wang et al., 2021]. Au contraire, nous proposons une approche d'ensemble combinant de multiples plongements afin de plonger différents aspects complémentaires des relations du graphe de connaissance. Deuxièmement, les approches existantes évaluent uniquement leurs prédictions par rapport aux médicaments en cours d'essai et ne fournissent aucune évaluation moléculaire de leurs prédictions. Nous fournissons une évaluation moléculaire de nos prédictions par comparaison avec des ligands connus des cibles COVID-19. Enfin,

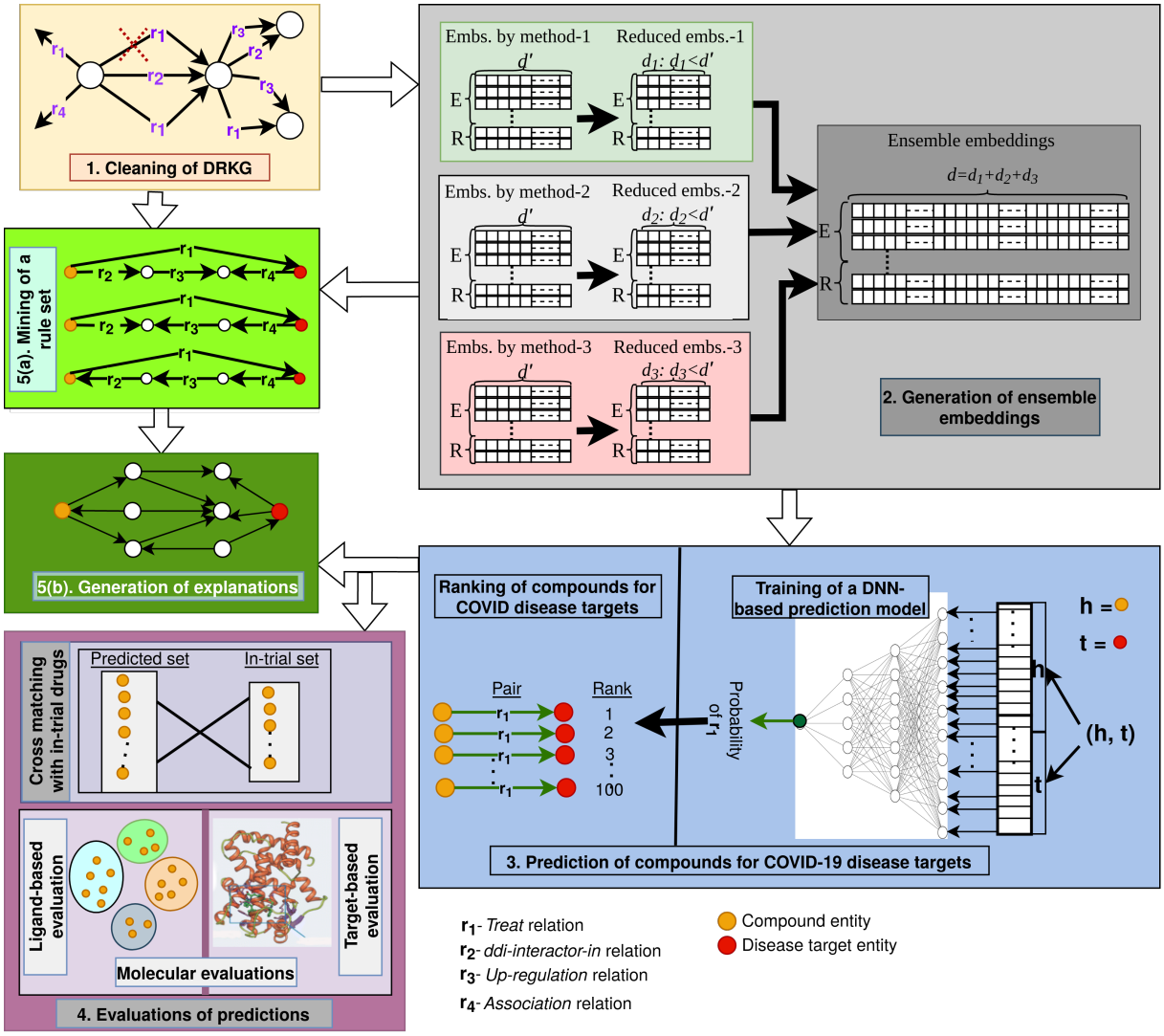


FIGURE 3.4 – Flux de travail global de l'étude ; les principales étapes sont numérotées. Étape 1 (boîte jaune) ; nettoyage d'un graphe de connaissances pour le repositionnement des médicaments centré sur le COVID-19 (DRKG). Étape 2 (boîte grise) : apprentissage d'ensembles de plongements de haute qualité. Étape 3 (boîte bleue) : prédire et classer les médicaments potentiels pour les cibles de la maladie COVID-19. Étape 4 (boîte violette) : évaluation des 100 meilleurs composés sur la base du croisement avec des médicaments en cours d'essai (panneau supérieur) et évaluation moléculaire des composés ciblant la protéine nsp13 du SRAS-CoV-2 (panneaux inférieurs). Étape 5a (encadré vert clair) : fouille d'un ensemble de règles d'explication à partir de DRKG. Étape 5b (encadré vert foncé) : extraction des chemins explicatifs instanciant les règles pour des paires d'intérêt données (Composés, Maladie). [Islam et al., 2023]

alors que les explications des prédictions dérivées du KG manquent dans la plupart des approches existantes, nous fournissons des explications basées sur des règles extraites du graphe de connaissance et cela contribue à améliorer la fiabilité de nos prédictions.

3.5 Résultats

3.5.1 Évaluation de l'échantillonnage négatif SNS

Pour évaluer l'efficacité de l'échantillonnage SNS, nous connectons SNS à un modèle de plongement géométrique de graphes de connaissances. Les modèles de plongement géométrique sont généralement classés en deux catégories principales : (1) les modèles translationnels (translational models) et (2) les modèles de correspondance sémantique (semantic matching models) [Wang et al., 2017]. Les modèles translationnels considèrent chaque relation comme une translation dans l'espace de représentation : ajouter une relation à la tête donne une position proche de la queue [Bordes et al., 2013]. D'autre part, les modèles de correspondance sémantique définissent la plausibilité d'un triplet en faisant correspondre les représentations des entités et des relations incarnées dans leurs plongements [Wang et al., 2017]. En ce qui concerne l'évaluation de la méthode SNS, nous avons choisi TransH [Wang et al., 2014] pour représenter les modèles translationnels et *DistMult* [Yang et al., 2015] pour représenter le modèle de correspondance sémantique car ce sont des bases de référence populaires pour la tâche de prédiction de lien dans les graphes de connaissances. Chacun des modèles est évalué en combinaison avec trois méthodes d'échantillonnage (random-uniform, modèle basé sur les GANs, self-adversarial et NSCaching) et avec la méthode SNS. Pour la méthode basée sur les GANs, nous choisissons KBGAN car il s'agit du seul échantillonnage basé sur le GAN dont la mise en oeuvre est accessible au public.

Dans nos expérimentations, nous utilisons cinq jeux de données/graphes de connaissances de référence largement utilisés : FB15K, FB15K-237, WN18, WN18RR, YAGO3-10 pour la tâche de prédiction de lien [Bordes et al., 2013, Cai and Wang, 2018, Zhang et al., 2019]. WN18, WN18RR ont été dérivés de WordNet, FB15K, FB15K-237 ont été extraits de Freebase et YAGO3-10 a été extrait des connaissances YAGO. Le jeu de données Family est un petit graphe de connaissances, où les relations sont les relations familiales entre les personnes [Kok and Domingos, 2007, Sadeghian et al., 2019].

Nous construisons les modèles de KGE à partir de zéro pour chacune des méthodes d'échantillonnage, aléatoire-uniforme, KBGAN, NSCaching, SNS. Pour le paramétrage, nous suivons les recommandations de l'article original ([Zhang et al., 2019] pour NSCaching, [Cai and Wang, 2018] pour KBGAN). Les performances de la méthode SNS sont présen-

tées dans le tableau 3.1. Compte tenu du modèle translationnel (TransH), la méthode d'échantillonnage SNS proposée montre sans aucun doute les meilleures métriques de prédiction parmi toutes les méthodes d'échantillonnage négatives par rapport à la plupart des métriques de prédiction dans la plupart des ensembles de données KG. Dans les ensembles de données WordNet (WN18RR, WN18), NSCaching est la deuxième meilleure méthode et les performances sont améliorées de 2 à 5% avec l'échantillonnage SNS en tenant compte des métriques Hit@k (voir Section 1.3.2) lorsqu'elles sont utilisées avec TransH. Nous trouvons les meilleurs résultats de l'échantillonnage SNS pour l'ensemble de données FB15K où les scores Hit@k sont améliorés de 5 à 8% par rapport à la deuxième meilleure méthode d'échantillonnage NSCaching. Pour les deux autres ensembles de données, SNS reste également le meilleur ou le deuxième meilleur en ce qui concerne presque toutes les métriques. Lorsque les méthodes d'échantillonnage sont utilisées avec le modèle *DistMult*, les scores hit@10 et hit@3 chutent dans tous les ensembles de données. L'entrainement du modèle avec plus d'époques pourrait améliorer les métriques. Cependant, notre intention n'est pas de comparer différents modèles de prédiction, mais plutôt différentes méthodes d'échantillonnage.

3.5.2 Évaluation de l'explicabilité

Nous calculons les performances de prédiction de notre méthode de fouille de règles basée sur les plongements appris par notre méthode SNS et nos stratégies d'échantillonnage aléatoire. Le tableau 3.2 présente les scores de rappel avec leurs nombres moyens, minimum et maximum pour les jeux de données Family, WN18RR et FB15K-237. Le tableau 3.3 répertorie les scores des métriques de qualité avec leurs nombres moyens, minimum et maximum pour les mêmes ensembles de données.

Dans l'ensemble, notre méthode de fouille de règles génère plus de règles lorsque les plongements sont générés à l'aide de *DistMult* et avec SNS comme méthode d'échantillonnage négatif. Mais nous ne voyons aucune différence significative entre les scores de performance des ensembles de règles générées. Nous voyons que le nombre moyen de règles extraites par relation est le plus élevé pour les graphes denses FB15K-237 et le plus bas pour les graphes éparses WN18RR. Ceci est attendu car une valeur élevée de degré moyen des nœuds d'un graphe conduit à générer plus de chemins entre une paire de nœuds et donc plus de règles pour décrire la relation entre la

TABLE 3.1 – Résultats de prédiction de liens (LP) : MRR et Hit@z de différentes méthodes d'échantillonnage négatif (NS) avec différentes méthodes de KGE. Random et Self-Adv représentent respectivement les méthodes d'échantillonnage aléatoire-uniforme et "self-adversarial". Les meilleures et deuxièmes meilleures métriques pour chaque méthode d'échantillonnage avec chaque méthode de plongement sont marquées en gras et soulignées, respectivement. [Islam et al., 2022]

KGE model	**NS method**	**WN18RR**				**WN18**			
		MRR	hit@10	hit@3	hit@1	MRR	hit@10	hit@3	hit@1
TransH	Random	0.1520	32.27	23.72	0.11	0.3199	79.43	64.25	11.85
	NSCaching	0.1713	40.68	31.43	0.93	0.4171	88.65	74.05	17.48
	KBGAN	0.1708	40.08	29.35	0.10	0.4183	87.34	73.67	18.09
	Self-Adv	0.1709	41.18	31.16	0.89	0.4191	89.48	75.84	7.08
	SNS	**0.1852**	**43.04**	**33.82**	**1.80**	**0.448**	**91.47**	**79.30**	**19.28**
DistMult	Random	0.1918	32.16	23.88	14.22	0.3453	52.82	37.33	25.75
	NSCaching	0.2262	37.37	**29.11**	17.84	0.3772	56.85	**42.18**	29.04
	KBGAN	0.2285	33.42	27.23	17.34	0.3791	57.28	41.97	28.39
	Self-Adv	0.2279	36.23	26.34	17.05	**0.3940**	**58.43**	41.31	31.29
	SNS	**0.2333**	**37.83**	25.12	**18.49**	0.3931	56.46	42.13	**31.38**
		FB15K237				**FB15K**			
TransH	Random	0.1988	36.68	22.50	11.50	0.3115	52.88	36.68	19.58
	NSCaching	0.2476	40.39	26.59	**17.33**	0.3926	61.22	44.99	25.50
	KBGAN	0.2162	40.58	23.52	15.23	0.3228	53.67	38.34	20.52
	Self-Adv	0.2350	41.01	26.49	14.73	0.3883	61.09	44.05	25.74
	SNS	**0.2514**	**42.90**	**29.44**	15.18	**0.4360**	**66.72**	**51.52**	**30.58**
DistMult	Random	0.1918	31.82	20.71	12.96	0.2188	33.25	21.84	12.95
	NSCaching	0.2205	34.87	25.69	15.72	0.2327	39.89	27.41	16.19
	KBGAN	0.2282	36.23	**27.23**	13.02	0.2203	35.54	23.12	15.92
	Self-Adv	0.2400	37.09	25.82	17.34	0.2493	39.80	28.38	18.53
	SNS	**0.2471**	**38.18**	26.97	**17.80**	**0.2937**	**45.62**	**32.66**	**20.79**
		YAGO3-10				**Family**			
TransH	Random	0.0850	18.96	9.05	1.24	0.3164	88.77	48.67	3.43
	KBGAN	0.1467	26.08	16.03	8.34	0.3742	90.21	57.12	8.63
	Self-Adv	0.1443	26.46	**17.34**	**9.92**	0.3510	89.58	56.36	8.08
	NSCaching	0.1431	**26.76**	15.97	7.49	0.3434	90.34	55.04	4.65
	SNS	**0.1488**	26.72	16.23	8.87	**0.4146**	**92.97**	**62.39**	**13.12**
DistMult	Random	0.0533	10.59	5.44	2.35	0.5002	77.36	60.14	37.45
	KBGAN	0.0712	13.98	7.79	3.19	0.5120	78.01	60.75	38.01
	Self-Adv	0.0868	16.04	8.74	5.07	0.5122	78.59	60.95	**38.33**
	NSCaching	**0.0875**	14.22	**8.86**	**5.64**	**0.5274**	78.55	**61.34**	38.07
	SNS	0.0804	**16.72**	8.39	4.98	0.5190	**79.32**	61.02	37.98

paire de nœuds. Le nombre élevé de règles pour les jeux de données Family et FB15K-237 donne des scores de rappel élevés dans ces deux graphes de connaissances. En regardant les scores de rappel minimum, nous voyons que les jeux de données, à l'exception de Family, donnent un score minimum de 0. En étudiant le rappel par rapport aux relations, nous constatons que la qualité des règles extraites pour une relation avec un faible nombre d'ensembles d'apprentissage est très faible et dans certains cas, aucune règle n'est générée pour quelques relations de ce type. Cette situation affecte les scores de performance ainsi que la qualité globale de l'ensemble de règles d'une relation. Dans le tableau 3.3, nous voyons les pires métriques de qualité des règles pour WN18RR. Le degré moyen de WN18RR est faible, ce qui réduit le nombre de chemins entre deux entités pour générer des règles. C'est la raison la plus probable est que notre méthode extrait les

règles de longueur 2 de faible qualité pour WN18RR.

TABLE 3.2 – Métriques de performance des règles : #Rules représente le nombre total de règles extraites de longueur 2. La métrique de rappel est donnée dans le format (score minimum, score moyen, score maximum) sont calculés sur toutes les relations dans un graphe de connaissances. 'DistMult' est testée avec deux méthodes d'échantillonnage différentes : SNS et la méthode aléatoire (Random). [Islam et al., 2022]

KGs	Random		SNS	
	#Rule	Recall (min.,avg., max., std)	#Rules	Recall (min.,avg., max.,std)
Family	235	(0.569, 0.908, 1.0, 0.157)	242	(0.586, 0.910, 1.0, 0.150)
WN18RR	69	(0.0, 0.184, 0.487, 0.162)	76	(0.0, 0.185, 0.487, 0.164)
FB15K-237	8490	(0.0, 0.684, 1.0, , 0.262)	8559	(0.0, 0.689, 1.0, 0.258)

TABLE 3.3 – Scores de qualité des règles : HC et SC désignent respectivement la couverture de la tête et les mesures de confiance standard. Les métriques sont données au format (score minimum, score moyen, score maximum) sont calculées sur toutes les relations d'un graphe de connaissances. 'DistMult' est testée avec deux méthodes d'échantillonnage différentes : SNS, aléatoire (Random). [Islam et al., 2022]

KGs	Random		SNS	
	HC(min., avg., max.)	SC(min., avg., max.)	HC(min., avg., max.)	SC(min., avg., max.)
Family	(0.135, 0.208, 0.241)	(0.325, 0.523, 0.674)	(0.119, 0.205, 0.254)	(0.285, 0.531, 0.695)
WN18RR	(0.0, 0.0337, 0.098)	(0.0, 0.0569, 0.179)	(0.0, 0.0311, 0.0869)	(0.0, 0.0689, 0.1433)
FB15K-237	(0.0, 0.176, 0.969)	(0.0, 0.196, 0.905)	(0.0, 0.175, 0.969)	(0.0, 0.199, 0.905)

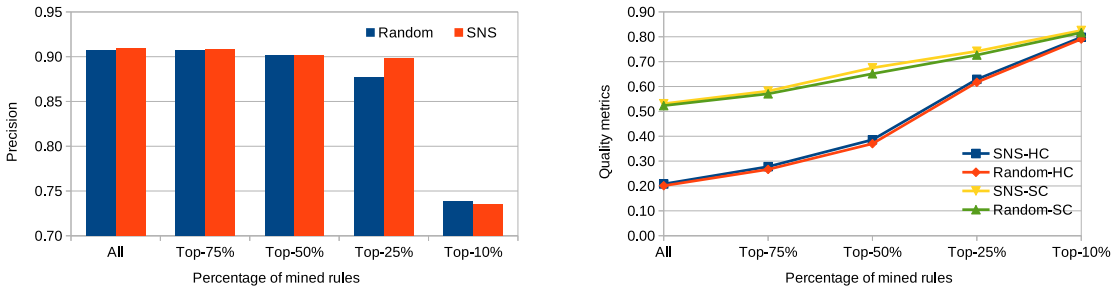
Dans la liste de nos priorités, nous cherchons à étudier comment la qualité des plongements affecte la qualité de l'ensemble de règles extraites ainsi que les performances de prédiction dans le jeu de données WN18RR. Nous avons choisi WN18RR, car les métriques de prédiction globales et les métriques de qualité des règles de ce jeu de données sont les plus basses. Nous sélectionnons les règles de longueur-2 extraites sur la base du plongement généré par *DistMult* avec l'échantillonnage SNS. Nous voyons que les résultats sont presque cohérents avec les résultats de prédiction par la méthode basée sur les plongements présentés dans le tableau 3.4. Nous voyons qu'aucune règle n'est extraite pour la relation similar_to car la relation a un nombre insuffisant de triplets dans l'ensemble d'apprentissage.

Pour voir comment les performances de fouille de règles changent avec différents pourcentages de règles, nous échantillonnons toutes les règles de longueur-2, les top-75%, les top-50%, les top-25% et les top-10% pour chaque relation dans le graphe Family (voir Figure 3.5). Les règles

TABLE 3.4 – Performances (relation-wise) des règles de longueur 2 extraites dans le jeu de données WN18RR : les plongements sont appris par *DistMult* avec un échantillonnage SNS. Les paramètres des méthodes de fouille de règles sont définis sur les valeurs par défaut, sauf que la longueur maximale de la règle est fixée à 2. Les relations avec des scores de rappel supérieurs à la moyenne (0,184) sont indiquées en gras. [Islam et al., 2022]

Relations	#Rule	Recall	HC	SC
hypernym	9	0.089	0.045	0.009
derivationally_related_form	12	0.224	0.087	0.019
instance_hypernym	5	0.197	0.017	0.028
also_see	7	0.214	0.048	0.040
member_meronym	6	0.099	0.013	0.016
synset_domain_topic_of	15	0.447	0.143	0.032
has_part	10	0.238	0.057	0.023
member_of_domain_usage	2	0.042	0.131	0.043
member_of_domain_region	2	0.00	0.142	0.015
verb_group	8	0.487	0.006	0.087
similar_to	0	0.000	0.00	0.00

d'une relation sont classées par ordre décroissant de leurs valeurs de support. Comme les règles de rang inférieur ne sont pas prises en compte, les règles de déclenchement de quelques triplets seront manquantes. En conséquence, on observe sur la Figure 3.5(a) que les scores de rappel pour les deux méthodes d'échantillonnage diminuent quand le nombre de règles diminue. En revanche, comme les règles de faible qualité sont supprimées, les scores moyens de qualité des règles s'améliorent (voir Figure 3.5(b)).



(a) Prediction performance

(b) Rule quality

FIGURE 3.5 – Changement des performances et de la qualité des prédictions avec différents pourcentages de règles extraites de longueur 2 dans le graphe Family. [Islam et al., 2022]

Dans le tableau 3.5, nous tabulons les 5 règles extraites les mieux classées pour la relation "Father" dans le graphe Family. En examinant la couverture de la tête et les scores de confiance

standard, nous constatons que notre méthode est capable de calculer de meilleurs classements pour les règles de meilleure qualité.

TABLE 3.5 – Top-5 des règles de longueur-3 pour la relation 'père' : les rangs sont calculés dans l'ordre décroissant des valeurs de support des règles. [Islam et al., 2022]

Rank	Rules	HC	SC
1	$husband(A,B) \wedge mother(B,C) -> father(A,C)$	0.765	0.882
2	$father(A,B) \wedge brother(B,C) -> father(A,C)$	0.599	0.407
3	$father(A,B) \wedge son(B,C) \wedge mother(C,D) -> father(A,D)$	0.550	0.298
4	$husband(A,B) \wedge mother(B,C) \wedge brother(C,D) -> father(A,D)$	0.546	0.386
5	$father(A,B) \wedge sister(B,C) -> father(A,C)$	0.524	0.420

3.5.3 Évaluation de la prédiction de médicaments pour le COVID-19

Le modèle de prédiction que nous avons proposé a été entraîné avec un ensemble de 261,080 paires d'entraînement de type *(Composé, Maladie)* et validé sur 5,800 paires de test. Nous trouvons une erreur quadratique moyenne (MSE ou *Mean Squared Error*) de 0,037 sur l'ensemble de test pour le modèle de prédiction formé. Le modèle formé est ensuite utilisé pour calculer les rangs des composés candidats en fonction de leurs valeurs de probabilité par rapport aux entités de la maladie COVID-19. Il est important de mentionner que les 100 meilleures prédictions ne correspondent pas à une cible unique mais plutôt à plusieurs cibles de la maladie COVID-19. Pour des raisons de simplicité, nous ne fournissons que les 20 composés les plus prédits dans le tableau 3.6. Nous fournissons également la meilleure cible de maladie pour chaque composé. Nous constatons que les classements des composés en cours d'essai pour le COVID-19 sont sensiblement améliorés par rapport aux approches de pointe (voir Tableau 3.6).

Pour évaluer l'efficacité de notre approche proposée de repositionnement des médicaments, il est important d'évaluer nos prédictions par rapport aux médicaments recommandés ou en cours d'essai. Nous effectuons deux types d'évaluation pour nos 100 composés les mieux classés : le croisement avec des médicaments en cours d'essai pour le COVID-19 et des évaluations moléculaires (uniquement pour la cible de la maladie SARS-CoV-2-nsp13).

Croisement avec des médicaments en cours d'essai

Lorsque nous avons croisé les 100 principaux composés avec les 31 composés en cours d'essai pour le COVID-19 [Islam et al., 2023], nous constatons que les 10 composés les mieux classés sont en fait des composés à l'essai pour le COVID-19 (voir Tableau 3.6). Nous trouvons 10/31 médicaments en cours d'essai dans les 100 meilleures prédictions de notre approche, ce qui est remarquable par rapport à la plupart des approches existantes.

TABLE 3.6 – Top 20 des médicaments classés : les composés en cours d'essai sont surlignés en bleu, - représente l'indisponibilité du rang d'un médicament, * met en évidence les médicaments en cours d'essai trouvés uniquement par notre approche, les cinq dernières colonnes donnent le rang ou la prédiction (✓) de composés par différentes approches. [Islam et al., 2023]

Compound	Disease target	Our approach	Tex-Graph[Kanatsoulis and Sidiropoulos, 2021]	TransE-DRKG[Ioannidis et al., 2020]	ENSIGN[Leggas et al., 2021]	PERM[Choudhary et al., 2021]
Dexamethasone	SARS-CoV-2-nsp6	1	1	4	✓	✓
Methylprednisolone	SARS-CoV-2-nsp6	2	6	16	✓	-
Ruxolitinib*	SARS-CoV-2-nsp13	3	-	-	-	-
Colchicine	SARS-CoV-2-nsp6	4	48	8	-	-
Thalidomide	SARS-CoV-2-nsp5_C145A	5	18	-	-	-
Chloroquine	SARS-CoV-2-nsp5_C145A	6	68	-	-	-
Azithromycin	SARS-CoV-2-nsp6	7	13	-	-	-
Losartan	SARS-CoV-2-nsp13	8	41	-	✓	-
Baricitinib*	SARS-CoV-2-nsp5_C145A	9	-	-	-	-
Hydroxychloroquine	SARS-CoV-2-nsp5_C145A	10	47	-	-	✓
Protirelin (DB09421)	SARS-CoV-2-nsp13	11	-	-	-	-
Telavancin (DB06402)	SARS-CoV-2-nsp13	12	-	-	-	-
Propiomazine (DB00777)	SARS-CoV-2-nsp14	13	-	-	-	-
Hydroxyzine (DB00557)	SARS-CoV-2-nsp13	14	-	-	-	-
Indinavir (DB00224)	SARS-CoV-2-nsp5_C145A	15	-	-	-	-
Nafcillin (DB00607)	SARS-CoV-2-nsp5_C145A	16	-	-	-	-
Bifonazole (DB04794)	SARS-CoV-2-nsp13	17	-	-	-	-
Obeticholic acid (DB05990)	SARS-CoV-2-nsp13	18	-	-	-	-
Meclizine (DB00737)	SARS-CoV-2-nsp13	19	-	-	-	-
Lovastatin (DB00227)	SARS-CoV-2-nsp6	20	-	-	-	-

Le plus grand nombre de composés (40) dans le top 100 correspond à la cible SARS-CoV-2-nsp13. Parmi ces 40 composés, 2 sont en essai et 38 sont nouveaux. Nous avons choisi de concentrer nos évaluations moléculaires sur ces nouveaux composés.

Évaluation moléculaire de composés pour SARS-CoV-2-nsp13

Pour les évaluations moléculaires, nous comparons nos 38 composés prédits avec 86 composés connus pour se lier à la cible SARS-CoV-2-nsp13 (38 de la littérature et 48 de la base de données PDB [Berman et al., 2000]). Nous fournissons des évaluations moléculaires basées sur des structures de ligand ou de cible et celles-ci sont nommées évaluations basées sur le ligand ou basées sur la cible, respectivement.

Évaluation basée sur les ligands. Cette évaluation consiste à regrouper les ligands prédits et connus en fonction de leur similarité structurelle mesurée avec le coefficient de Tanimoto. Nous trouvons un total de treize clusters. Parmi ces clusters, dix contiennent à la fois des ligands prédits et connus et leur contenu est répertorié dans le tableau 3.7. Nous fournissons également

le poids moléculaire (MW) de la sous-structure commune maximale (MCS) des ligands dans chaque groupe. Le premier cluster contient 10 ligands dont 6 sont connus de la littérature et 4 sont de nouveaux composés prédits. De plus, ce cluster affiche le MCS le plus élevé, révélant ainsi une bonne similarité moléculaire entre tous ces composés. Sur la base des valeurs de MCS qui affichent une diminution importante entre les clusters 6 et 7, on pourrait retenir tous les nouveaux composés des clusters 1 à 6 (*i.e.*, 18 composés) comme potentiels ligands intéressants pour la cible nsp13. Cet ensemble de 18 composés comprend le Fosinopril qui apparaîtra également dans l'évaluation basée sur les cibles.

Évaluation basée sur les cibles. Cette évaluation consiste à réaliser un docking (ou "amarrage") moléculaire des 38 ligands prédits et 86 connus dans le site actif de la structure nsp13 à l'aide du logiciel GOLD. Le docking moléculaire est une approche numérique de modélisation moléculaire. Elle cherche à identifier la meilleure orientation entre 2 molécules sur la base de leur coordonnées dans des états pris séparément. [Yang et al., 2022]. Le tableau 3.8 répertorie les 20 meilleurs ligands en ce qui concerne l'amarrage moléculaire. Il est important de mentionner que 4/38 ligands prédits sont présents dans cette liste, avec des scores d'amarrage supérieurs à 70. En particulier, Fosinopril se démarque et se classe en deuxième position, avec un score (78,86) très similaire au ligand de premier rang Diosmine (79.04), identifié comme ligand nsp13 par White *et al.* [White et al., 2020]. Les trois autres ligands prédits : Macitentan, Eprosartan et Dinoprostone sont classés en position 12 à 14, avec des scores allant de 70,76 à 71,76 malgré leur MW variable, excluant ainsi un effet de leur taille sur le nombre d'interactions avec la cible.

Le tableau 3.8 affiche également la meilleure cible de maladie pour chaque ligand et le score de probabilité des triplets formés entre les ligands et les cibles de maladie COVID-19, à l'exception de quatre d'entre eux (marqués d'un astérisque) qui ne sont pas présents dans DRKG. Lorsque la meilleure cible de la maladie n'est pas nsp13, les informations pour la deuxième meilleure cible sont fournies uniquement s'il s'agit de nsp13. Étonnamment, le meilleur ligand amarré Diosmine n'affiche pas nsp13 comme meilleure ou deuxième meilleure cible.

Nous avons ensuite analysé nos résultats d'amarrage en explorant les cartes d'interaction de nos quatre composés les mieux classés et en les comparant à Diosmine (ligand connu avec le meilleur score Gold), Ergotamine et Risperdal (ligands connus avec les deux scores de probabilité les plus élevés avec nsp13 dans DRKG) pour vérifier leurs similitudes de liaison. La Figure 3.6

TABLE 3.7 – Clustering des 100 meilleurs ligands prédits et connus (d'après la littérature et le PDB) : le nombre entre () donne un poids moléculaire entier. Les clusters sont numérotés dans l'ordre décroissant de leur poids maximum de sous-structure commune (MCS). Les ligands connus de la base de données PDB sont désignés par leur abréviation PDB. Les noms complets correspondants sont fournis dans [Islam et al., 2023]

No	**Predicted ligands**	**Known ligands**	**MW** (g/mol) for MCS
1	Fosinopril(563), Griseofulvin(352), Telavancin(1755), Ridaforolimus(990)	Simeprevir(749), Dihydroergotamine(583), Paritaprevir(765), Ergoloid(611), Grazoprevir(766), Ergotamine(581)	120
2	Protirelin(362), Teriparatide(4117), Tiagabine(375)	NUA(193), HR5(207), VWM(187)	105
3	Binimetinib(441), Niflumic_acid(282), Moxifloxacin(401)	Picrasidine N(490), Irinotecan(586), Netupitant(578), Lumacaftor(452), Bananin(327), Picrasidine M(490), Nilotinib(529), Zelboraf(489)	98
4	Mesoridazine(386), Perphenazine(403), Oxcarbazepine(252), Tizanidine(253)	SSYA10-001(308), NY7(194), VVD(197), VW7(204), S7G(190), UVA(185), N0E(241), NZG(197), JHJ(243), LJA(193), EJQ(222), VXD(198)	93
5	Meclizine(390), Flunarizine(404), Remoxipride(371), Hydroxyzine(374)	Lifitegrast(615), Emend(534), UR7(203), JOV(227), VWD(200), NX7(211), VWA(153), UQS(191), UXG(239), VW4(199)	92
6	Darifenacin(426), Tetrabenazine(317), Oxybutynin(357), Ritodrine(287), Oxymorphone(301), Naloxone(327), Hydrocodone(299), Tofisopam(382)	Tubocurarine(609), Cepharanthine(606), Differin(412), Isorhoeadine(383), Epiexcelsin(414), Enjuvia(350), Homovanillic acid(182), K2P(206), VXG(233), VVY(205), STV(243), VW1(190)	89
7	Lamotrigine(256)	K34(152), JG4(150)	67
8	Carisoprodol(260)	Clavulanic acid(199), Acetylcysteine(163), VWV(221)	59
9	Macitentan(588), Famotidine(337), Eprosartan(424)	Cefoperazone(645), Cefpiramide(612), Risperdal(410), Cordycepin(251), Pritelivir(402), Dpnh(665), VX4(224)	50
10	Risedronic acid(283), Cinchocaine(343), Bifonazole(310)	RYM(233), NYV(189), VWJ(175), O2A(174), UVJ(203), MUK(199), S7J(191), UJK(203), VWG(188)	50

montre la superposition des ligands (4 prédits contre 3 connus) extraits des meilleures poses d'amarrage correspondantes et le tableau 3.9 compare la liste des résidus nsp13 concernés par

TABLE 3.8 – Liste des 20 meilleurs ligands ancrés (docked) classés selon le Gold Score décroissant. * signifie que le ligand n'est pas présent dans DRKG. Les résultats pour les ligands de nos prédictions sont mis en évidence en couleur bleu clair. Les valeurs de prédiction dans DRKG sont également indiquées avec la meilleure cible de maladie et le score de probabilité correspondant. Lorsque la meilleure cible n'est pas nsp13, la deuxième meilleure cible et son score de probabilité ne sont indiqués que s'il s'agit de nsp13. [Islam et al., 2023]

Docking rank	Ligand name	Gold score	MW (g/mol)	Best predicted target	Predicted probability score	Predicted rank	Reference
1	Diosmin	79.04	608.5	nsp9	0.169	6091	White *et al.* (2020) [White et al., 2020]
2	Fosinopril	78.86	563.7	nsp13	0.964	29	This study
3	Nilotinib	76.55	529.5	nps6/nsp13	0.860/0.796	101	White *et al.* (2020) [White et al., 2020]
4	Chromone-4c*	76.17	417.4	NA	NA	NA	Perez-Lemus *et al.* (2022) [Perez-Lemus et al., 2022]
5	Dpnh	76.04	665.4	nsp9	0.146	6361	White *et al.* (2020) [White et al., 2020]
6	Cromolyn	75.7	468.4	nsp13	0.619	1751	White *et al.* (2020) [White et al., 2020]
7	Picrasidine_N*	74.55	490.5	NA	NA	NA	Vivel-Ananth *et al.* (2022) [Vivek-Ananth et al., 2022]
8	Picrasidine_M*	74.52	490.5	NA	NA	NA	Vivel-Ananth *et al.* (2022) [Vivek-Ananth et al., 2022]
9	Dihydroergotamine	74.13	583.7	nsp13	0.796	475	White *et al.* (2020) [White et al., 2020]
10	Ergotamine	72.83	581.7	nsp13	0.805	361	White *et al.* (2020) [White et al., 2020]
11	Simeprevir	72.68	749.9	nsp6/nsp13	0.537/0.139	2076	Gurung (2020) [Gurung, 2020]
12	Macitentan	71.76	588.3	nsp13	0.95	49	This study
13	Eprosartan	71.33	424.5	nsp13	0.878	71	This study
14	Dinoprostone	70.76	352.5	nsp13	0.959	38	This study
15	Cefoperazone	70.24	645.7	nsp13	0.599	1816	White *et al.* (2020) [White et al., 2020]
16	Scutellarin*	70.19	462.4	NA	NA	NA	Gurung (2020) [Gurung, 2020]
17	Irinotecan	70.11	586.7	nsp6/nsp13	0.824/0.796	249	White *et al.* (2020) [White et al., 2020]
18	Paritaprevir	69.61	765.9	orf8	0.099	6990	Gurung (2020) [Gurung, 2020]
19	Risperdal	68.76	410.5	nsp6/nsp13	0.807/0.807	349	White *et al.* (2020) [White et al., 2020]
20	Ergoloid	68.59	611.7	nsp13	0.584	1888	White *et al.* (2020) [White et al., 2020]

des interactions avec tous ces ligands.

TABLE 3.9 – Liste des résidus d'acides aminés dans la structure nsp13 qui interagissent avec les ligands répertoriés (les ligands prédits sont surlignés en bleu clair, les autres correspondent à des ligands connus). Les ligands sont classés comme dans le tableau 3.8. L'étiquette et la position des résidus sont en gras lorsqu'elles correspondent à des résidus délimitant le site de liaison à l'ATP. Toutes les cartes d'interaction sont disponibles dans [Islam et al., 2023].

Ligands	GLY285	**GLY287**	**LYS288**	**SER289**	**HIS290**	**LYS320**	**GLU375**	**ARG442**	GLY538	GLU540
Diosmin	X	X	X	X	X	X		X	X	X
Fosinopril	X	X	X	X		X	X	X		X
Ergotamine	X		X	X	X	X	X	X	X	
Eprosartan			X	X		X				
Macitentan					X	X		X		
Dinoprostone	X	X	X	X	X	X		X		
Risperdal			X	X	X	X	X		X	X

Ces données montrent que les composés prédits partagent avec les ligands connus plusieurs éléments structuraux et deux d'entre eux (Fosinopril et Dinoprostone) interagissent avec 6/7

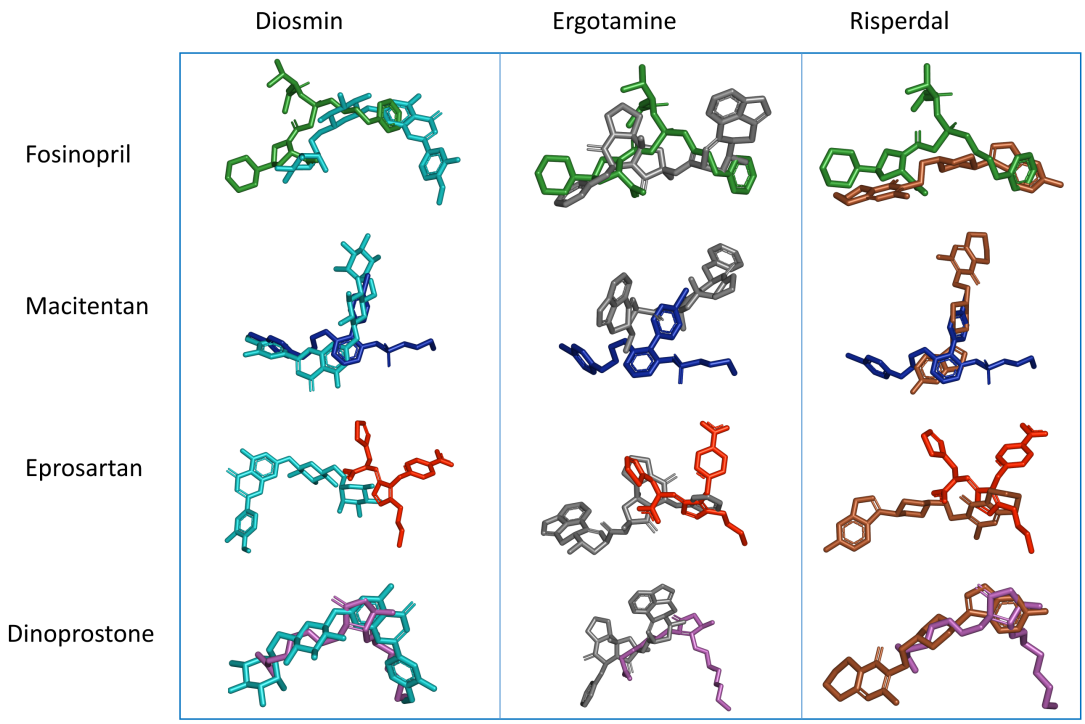


FIGURE 3.6 – Superposition des poses de liaison avec la cible nsp13 pour Fosinopril (en vert), Macitentan (en bleu foncé), Eprosartan (en rouge) et Dinoprostone (en magenta) contre Diosmine (en bleu clair), Ergotaminine (en gris) et Risperdal (en marron) . Les ligands le long de l'axe vertical sont les ligands prédits, tandis que ceux le long de l'axe horizontal sont les ligands connus. [Islam et al., 2023]

résidus d'acides aminés du site actif nsp13.

3.6 Discussion et conclusions

Bien que nos approches de prédiction de liens fournissent des résultats satisfaisants pour la prédiction de liens explicables dans les graphes de connaissances, nous identifions quelques limitations ainsi que des perspectives intéressantes à explorer. En particulier, nous comptons étudier l'utilisation des techniques d'apprentissage profond telles que les réseaux de neurones convolutifs et les réseaux de neurones récurrents pour la prédiction de liens dans les graphes de connaissances. Il est important de mentionner que la modélisation des relations complexes entre les entités et les concepts peut être difficile, et les modèles de prédiction de liens peuvent être sensibles aux erreurs de données et au sur-apprentissage. Il y a donc encore beaucoup de travail à faire pour améliorer les performances et la robustesse des approches de prédiction de liens dans

les graphes de connaissances.

Comme discuté dans la Section 3.4, nous avons appliqué des approches de prédiction de liens pour le repositionnement de médicaments en identifiant des associations entre les médicaments et les maladies qui n'ont pas encore été explorées. Cependant, ces approches de prédiction de liens pour le repositionnement de médicaments sont confrontées à plusieurs défis :

— Données insuffisantes : Les données sur les médicaments et les maladies sont souvent limitées et fragmentaires, ce qui peut rendre difficile la prédiction de nouvelles associations entre les médicaments et les maladies.

— Variabilité des données : Les données sur les médicaments et les maladies peuvent être très variables, car elles proviennent de différentes sources et peuvent avoir été mesurées de différentes manières. Cela peut rendre difficile la comparaison et l'intégration des données.

— Complexité des relations : Les relations entre les médicaments et les maladies peuvent être très complexes et impliquer de nombreux facteurs interdépendants. Cela peut rendre difficile la modélisation de ces relations et la prédiction de nouvelles associations.

— Hétérogénéité des populations : Les populations de patients peuvent varier considérablement selon les maladies, les groupes d'âge et les origines géographiques. Cela peut rendre difficile la généralisation des résultats et la prédiction de l'efficacité d'un médicament dans une population donnée.

— Validation expérimentale : Les prédictions doivent être validées expérimentalement avant de pouvoir être utilisées pour le repositionnement de médicaments. Cependant, la validation expérimentale peut être coûteuse et prendre du temps.

Deuxième partie

Calcul distribué et grands graphes complexes

Chapitre 4

Approches distribuées pour l'analyse de grands graphes dynamiques

4.1 Introduction

Au cours de la dernière décennie, le domaine du traitement distribué des graphes à grande échelle a attiré une attention considérable [Inoubli et al., 2018] [Aridhi and Mephu, 2016] [Inoubli et al., 2022]. Cette attention a été motivée non seulement par la taille croissante des données de graphes, mais aussi par leur grand nombre d'applications. Dans ce contexte, de nombreux systèmes de traitement de graphes distribués/parallèles ont été proposés, tels que Pregel [Malewicz et al., 2010], Ray [Moritz et al., 2018b], GraphLab [Low et al., 2012] et Trinity [Shao et al., 2013]. Ces systèmes peuvent être divisés en trois catégories : (1) approches centrées sur les sommets (ou *Vertex-centric*), (2) approches centrées sur les blocs (ou *Block-centric*) et (3) approches de type *Gather-Apply-Scatter* (GAS) [Xin et al., 2013]. Les approches centrées sur les sommets divisent les graphes d'entrée en partitions et utilisent un modèle de programmation "Think like a vertex" pour prendre en charge le calcul itératif des graphes [Malewicz et al., 2010] [Tian et al., 2013]. Chaque sommet correspond à un processus, et des messages sont échangés entre les sommets. Dans les approches centrées sur les blocs [Yan et al., 2014], l'unité de calcul est un bloc (un sous-graphe connexe du graphe) et les échanges de messages se produisent entre les blocs. Les approches centrées sur les sommets se sont avérées utiles pour de nombreux al-

gorithmes de graphes. Cependant, ils ne fonctionnent pas toujours efficacement, car ils ignorent les informations vitales sur les partitions de graphe, qui représentent des sous-graphes réels du graphe d'entrée d'origine, au lieu d'une collection de sommets non liés. Les approches de type GAS sont similaires aux approches basées sur les sommets et le programme qui s'exécute dans chaque sommet à chaque itération est divisé en trois sous-fonctions : *gather*, apply et *scatter*. Dans la fonction *gather*, le sommet courant obtient des informations de ses sommets voisins et éventuellement les agrège en une seule valeur σ. Dans la fonction *apply*, l'état du sommet est mis à jour en fonction de la valeur σ, et probablement des caractéristiques spécifiques des sommets voisins. Enfin, dans la fonction *scatter*, chaque sommet partage son nouvel état avec ses sommets voisins. Ensuite, chaque programme implémenté doit être modélisé comme trois fonctions qui manipulent les valeurs σ afin de changer l'état du sommet ou d'arrêter le programme.

Il est important de mentionner que les systèmes présentés ci-dessus ne traitent que des graphes statiques et ne considèrent pas la question du traitement des graphes évolutifs et dynamiques. Durant ces dernières années, nous nous sommes intéressés non seulement à la distribution de certaines tâches sur des graphes simples et/ou complexes à grande échelle mais aussi à la proposition de plateformes génériques pour le traitement de grands graphes dynamiques. Dans ce chapitre, je présente d'abord nos travaux d'état de l'art sur les plateformes de calcul distribué et je me focaliserai sur les plateformes spécifiquement conçues pour le traitement et l'analyse de grands graphes. Ensuite, je présente nos contributions pour le calcul distribué et le clustering incrémental dans des grands graphes dynamiques.

4.2 Plateformes de calcul distribué : état de l'art

L'étude des approches d'analyse de grands graphes est essentielle pour exploiter tout le potentiel des données massives représentées sous forme de graphes. Cela permet de découvrir des connaissances, de prendre des décisions éclairées et de résoudre des problèmes complexes dans divers domaines. Dans ce contexte, nous avons mené une étude comparative des plateformes existantes d'analyse de données de type graphe [Aridhi and Mephu, 2016] [Inoubli, 2021] et d'analyse de gros volumes de données ou de *Big Data* en général [Inoubli et al., 2018].

Framework	Programmi model	Distributed /Centralized	Communication	Dynamic/Static graph
Pregel	Vertex-centric	Distributed	Message passing	Static
GraphX	GAS	Distributed	Shared memory	Static
BLADYG	Block-Centric	Distributed	Hybrid	Dynamic
Giraph	Vertex-centric	Distributed	Message passing	Static
GraphIn	I-GAS	Centralized	Message passing	Dynamic
GraphMat	GAS	Centralized	Message passing	Static
PowerGraph	GAS	Distributed	Shared memory	Static
GraphLab	GAS	Distributed	Shared memory	Static
Blogel	Block-Centric	Distributed	Message passing	Static
Mezan	Vertex-centric	Distributed	Message passing	Static
GPS	Vertex-centric	Distributed	Message passing	Static
Trinity	GAS	Distributed	Shared memory	Static
Hama	Vertex-centric	Distributed	Message passing	Static
Pegasus	MapReduce	Distributed	no communication	Static
Giraph++	Vertex-centric	Centralized	Message passing	Static
GRACE	Vertex-centric	Centralized	message passing	Static

TABLE 4.1 – Graph processing frameworks comparative study [Inoubli, 2021].

4.2.1 Catégorisation des plateformes de calcul distribué sur des graphes

Il existe plusieurs plateformes de calcul distribué spécifiquement conçues pour le traitement et l'analyse de grands graphes. Dans le contexte d'un travail d'analyse critique de ces plateformes, nous avons présenté dans [Aridhi and Mephu, 2016] et dans [Inoubli, 2021] les architectures et les spécificités des plateformes populaires d'analyse de grands graphes. Le tableau 4.1 présente quelques plateformes de traitement de graphes les plus populaires. Pour chaque plateforme, nous listons le modèle de programmation, la méthode de communication utilisée et si le framework permet de traiter des graphes dynamiques ou non.

Tous les frameworks présentés dans le tableau 4.1 visent à fournir une interface de programmation facile à utiliser pour l'analyse de graphes qui peut ensuite être exécutée en utilisant les

ressources fournies. Nous mentionnons que la plupart des frameworks de traitement de graphes permettent des algorithmes asynchrones et distribués. Il est aussi important de mentionner que la plupart des frameworks distribués de traitement de graphes présentés traitent des graphes statiques et seul BLADYG (notre proposition, voir Section 4.3) considère les graphes évolutifs et dynamiques.

Une description détaillée des plateformes décrites dans le Tableau 4.1 ainsi qu'une catégorisation de quelques approches de fouille de graphes distribuées selon l'entrée, la sortie de chaque approche et le modèle de programmation utilisé sont disponibles dans [Aridhi and Mephu, 2016] et [Inoubli, 2021].

4.2.2 Catégorisation des plateformes de Big Data

Dans le cadre du travail de thèse de Wissem Inoubli [Inoubli, 2021], nous avons commencé par une étude théorique et expérimentale des plateformes de Big Data. Nous présentons dans le tableau 4.2 une comparaison des plateformes d'analyse de gros volumes de données en fonction de leurs principales caractéristiques. Ces caractéristiques clés sont : (1) le modèle de programmation, (2) les langages de programmation pris en charge, (3) le type de sources de données et (4) la capacité de permettre un traitement itératif des données, (5) la compatibilité de la plateforme avec les bibliothèques d'apprentissage automatique et (6) la stratégie de tolérance aux pannes.

Comme indiqué dans le tableau 4.2, Hadoop, Flink et Storm utilisent le format clé-valeur pour représenter leurs données. Ceci est motivé par le fait que le format clé-valeur permet d'accéder à des données hétérogènes. Pour Spark, le concept de "Resilient Distributed Dataset " (RDD), une collection d'éléments partitionnés et distribués à travers les nœuds d'un cluster, et le format clé-valeur sont utilisés pour permettre un accès rapide aux données. Nous avons également classé les frameworks Big Data étudiés selon deux modes de fonctionnement : (1) mode batch/lot et (2) mode flux. Nous avons montré dans le tableau 4.2 que Hadoop traite les données en mode batch, alors que les autres frameworks permettent le mode de traitement en flux. En termes d'architecture physique, on constate que tous les frameworks étudiés sont déployés dans une architecture en cluster, et chaque framework utilise un gestionnaire de cluster spécifique. Notons que la plupart des frameworks utilisent YARN comme gestionnaire de cluster. D'un point de vue technique, nous mentionnons que tous les frameworks présentés fournissent des API pour

	Hadoop	Spark	Storm	Flink	Samza
Data format	Key-value	Key-value, RDD	Key-value	Key-value	Events
Processing mode	Batch	Batch and Stream	Stream	Batch and Stream	Stream
Data sources	HDFS	HDFS, DBMS and Kafka	HDFS, HBase and Kafka	Kafka, Kinesis, message queus, socket streams and files	Kafka
Programming model	Map and Reduce	Transformation and Action	Topology	Transformation	Map and Reduce
Supported programming languages	Java	Java, Scala and Python	Java	Java	Java
Cluster manager	YARN	Standalone, YARN and Mesos	YARN or Zookeeper	Zookeeper	YARN
Comments	Stores large data in HDFS	Gives several APIs to develop interactive applications	Suitable for real-time applications	Flink is an extension of MapReduce with graph methods	Based on Hadoop and Kafka
Iterative computation	Yes (by running multiple MapReduce jobs)	Yes	Yes	Yes	Yes
Interactive Mode	No	Yes	No	No	No
Machine learning compatibility	Mahout	SparkMLlib	Compatible with SAMOA API	FlinkML	Compatible with SAMOA API
Fault tolerance	Duplication feature	Recovery technique on the RDD objects	Checkpoints	Checkpoints	Data partitioning

TABLE 4.2 – Comparative study of popular Big Data frameworks. [Inoubli et al., 2018]

plusieurs langages de programmation comme Java, Scala et Python. Chaque framework fournit un ensemble de fonctions abstraites qui sont utilisées pour définir le calcul souhaité. Nous avons également présenté dans le tableau 4.2 le fait que la plateforme permet d'exécuter des algorithmes d'apprentissage automatique ou non. Nous remarquons que Spark et Flink fournissent leurs propres bibliothèques d'apprentissage automatique, tandis que les autres frameworks ont une certaine compatibilité avec d'autres outils, tels que SAMOA [5] [De Francisci Morales, 2013] pour Samza et Mahout [6] [Anil et al., 2020] pour Hadoop.

Comme le montre le tableau 4.2, l'importance de Spark réside dans sa capacité d'utiliser la mémoire centrale d'une façon efficace et ses capacités de traitement par micro-lots, en particulier dans les cas de traitements itératifs [Bajaber et al., 2016]. Nous remarquons aussi que Spark est

5. `https://samoa-project.net/`
6. `https://mahout.apache.org/`

connu pour être très rapide dans certains types d'applications en raison du concept de RDD.

Flink partage des similitudes et des caractéristiques avec Spark. Il offre de bonnes performances de traitement lorsqu'il s'agit de structures de données complexes comme les graphes. Bien qu'il existe d'autres solutions pour le traitement de graphes à grande échelle, Flink et Spark sont enrichis de librairies et outils spécifiques pour l'apprentissage automatique, l'analyse prédictive et l'analyse de flux de graphes.

Une étude expérimentale approfondie des plateformes décrites dans le tableau 4.2 ainsi qu'un ensemble de bonnes pratiques liées à l'utilisation de ces plateformes dans plusieurs domaines d'application sont disponibles dans [Inoubli et al., 2018]. Dans cette étude, nous avons comparé les performances des plateformes étudiées pour plusieurs types de données (graphes, tweets, textes, etc) et dans deux scénarios différents : (1) traitement par lot et (2) traitement des données en flux.

4.3 Approche générique pour le calcul distribué sur des grands graphes dynamiques

Dans ce travail de recherche, nous considérons les questions d'échelle et de dynamisme dans le cas des approches centrées sur les blocs [Yan et al., 2014]. En particulier, nous étudions les grands graphes connus par leur nature évolutive et décentralisée. Par exemple, la structure d'un grand réseau social change au fil du temps (par exemple, les utilisateurs établissent de nouvelles relations et communiquent avec différents amis). Nous présentons *BLADYG*[7], un framework centré sur les blocs qui aborde la question du dynamisme dans les graphes à grande échelle. *BLADYG* peut être utilisé non seulement pour calculer les propriétés communes de grands graphes, mais aussi pour maintenir les propriétés calculées lorsque de nouvelles arêtes et nœuds sont ajoutés ou supprimés dans le graphe. L'idée clé est d'éviter le re-calcul des propriétés du graphe à partir de zéro lorsque le graphe est mis à jour. *BLADYG* limite le calcul à un petit sous-graphe en fonction de la tâche considérée. Nous présentons un ensemble d'abstractions pour *BLADYG* qui peuvent être utilisées pour concevoir des algorithmes pour n'importe quelle tâche de graphes distribués.

7. `http://bladyg.loria.fr/`

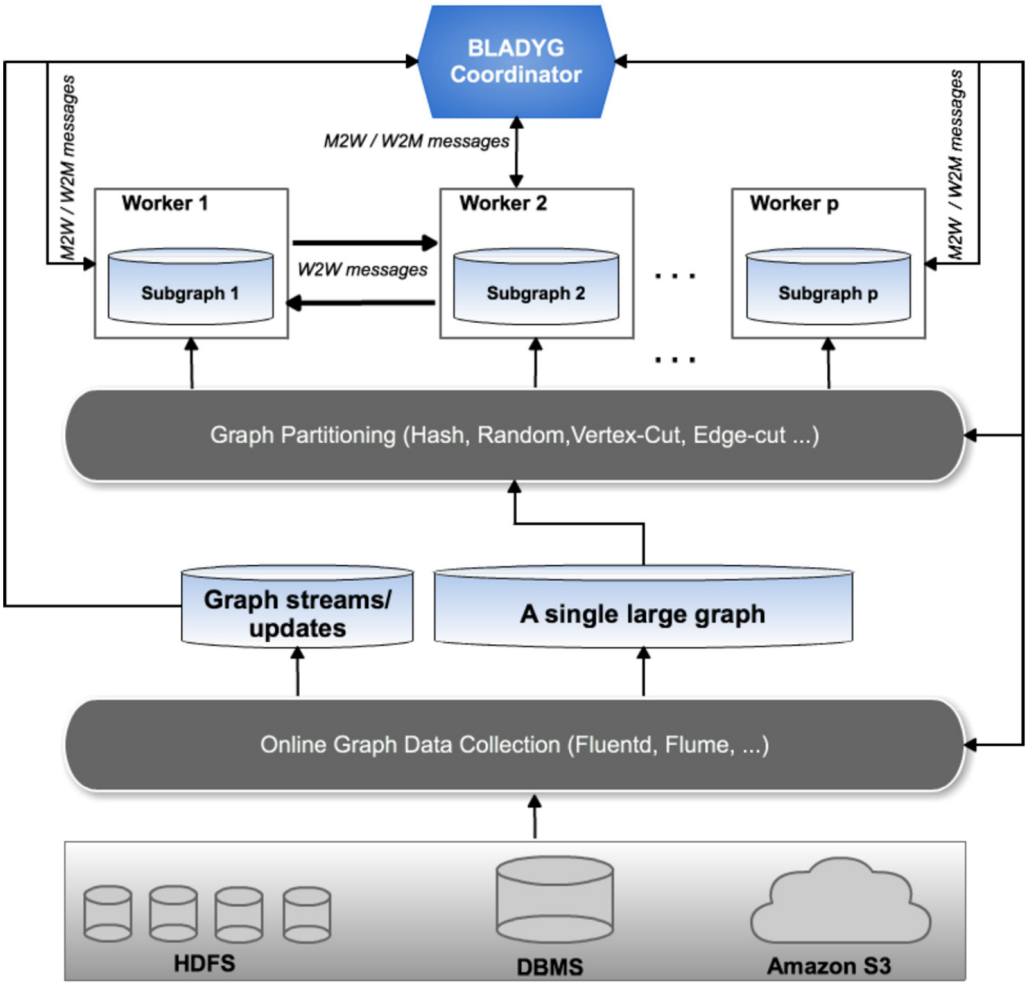


FIGURE 4.1 – Architecture de *BLADYG* [Aridhi et al., 2017]

La Figure 4.1 fournit une vue d'ensemble de l'architecture du framework *BLADYG*. Le calcul dans *BLADYG* commence en collectant les données du graphe à partir de diverses sources de données, notamment des fichiers locaux, des systèmes de fichiers distribués comme le Hadoop Distributed File System (HDFS) et Amazon Simple Storage Service (Amazon S3). Dans *BLADYG*, la collecte de données de graphes peut être effectuée à l'aide d'outils de collecte open source existants comme Kafka [Narkhede et al., 2017], NIFI [Blokdyk, 2020] [8], Flume [Chambers et al., 2010] et Fluentd [Singh, 2016]. Après avoir collecté les données du graphe, *BLADYG* partitionne le graphe d'entrée en plusieurs partitions, chacune d'entre elles étant affectée à une machine esclave (appelée aussi *worker*) différente. Chaque partition/bloc est un sous-graphe connexe du graphe d'en-

8. `https://nifi.apache.org/`

trée. Cette étape de partitionnement est effectuée par un worker qui prend en charge plusieurs types de techniques de partitionnement prédéfinies telles que le partitionnement par hachage, le partitionnement aléatoire, le egde-cut et le vertex-cut. Ces techniques de partitionnement sont présentées ainsi :

- Dans le partitionnement par hachage, les arêtes sont réparties sur les machines selon une fonction de hachage définie par l'utilisateur ;
- Dans le partitionnement aléatoire, les arêtes sont réparties sur les machines de manière aléatoire ;
- Dans le partitionnement vertex-cut, les arêtes sont uniformément réparties sur les machines dans le but de minimiser le nombre de sommets répliqués ;
- Dans le partitionnement edge-cut, les sommets d'un graphe sont divisés en clusters disjoints de taille presque égale, tandis que le nombre d'arêtes, qui s'étendent sur des clusters séparés, est minimum.

En plus des techniques de partitionnement fournies, les utilisateurs de *BLADYG* peuvent déployer des techniques de partitionnement de graphe existantes, notamment Metis [Karypis and Kumar, 1998a] et JaBeJa [Karypis and Kumar, 1998b]. Les utilisateurs de *BLADYG* peuvent également implémenter leurs propres méthodes de partitionnement. Il est important de mentionner que *BLADYG* permet de traiter de grands graphes déjà répartis sur un ensemble de machines. Ceci est motivé par le fait que la majorité des grands graphes existants sont déjà stockés de manière distribuée, soit parce qu'ils ne peuvent pas être stockés sur une seule machine en raison de leur taille, soit parce qu'ils sont traités et analysés avec des techniques décentralisées, qui les obligent à être répartis entre un ensemble de machines. Chaque machine worker charge son bloc et effectue des calculs locaux et distants, et ensuite l'état des blocs est mis à jour. La machine master orchestre l'exécution de *BLADYG* afin de gérer les mises à jour sur les données d'entrée. En fonction de l'algorithme de graphe, le master construit un plan d'exécution qui consiste en une liste ordonnée de calculs locaux et distants à exécuter par les workers.

Chaque worker effectue deux types d'opérations, comme suit :

1. Calcul *intra-bloc* : dans ce cas, le worker effectue un calcul local sur son bloc (partition)

associé et modifie soit l'état du bloc et/ou les états des nœuds à l'intérieur du bloc.

2. Calcul *inter-blocs* : dans ce cas, le worker demande aux workers distants de faire le calcul et après avoir reçu les résultats, il met à jour le statut de son bloc associé.

Le framework *BLADYG* fonctionne selon trois modes de calcul. En mode Master2Worker(M2W) et mode Worker2Master(W2M), les échanges de messages entre la machine maître (appelée aussi *master*) et tous les machines workers sont autorisées. Le maître/master utilise ce mode pour demander à un worker distant de rechercher des nœuds candidats, c'est-à-dire des nœuds qui doivent être mis à jour en fonction de la tâche considérée. Le worker utilise ce mode pour envoyer l'ensemble des nœuds candidats calculés au maître. En mode Worker2Worker (W2W), les échanges de messages entre workers sont autorisés. Les workers utilisent ce mode afin de propager la recherche de nœuds candidats à un ou plusieurs workers distants. En mode local, seul le calcul local est autorisé. Ce mode est utilisé par le worker/maître pour effectuer des calculs locaux.

Un calcul *BLADYG* typique consiste en : (1) un graphe d'entrée, (2) un ensemble de modifications/mises à jour, (3) une séquence d'opérations de worker ou de maître, et (4) une sortie.

4.4 Clustering distribué et incrémental des grands graphes dynamiques

Dans ce travail, nous nous sommes intéressés au clustering incrémental de graphes à grande échelle. Nous présentons *DISCAN*, une méthode distribuée et incrémentale de clustering de graphes basée sur les algorithmes de clustering structurel et en particulier l'algorithme *SCAN* décrit dans la Section 1.3.2. D'un point de vue architectural, *DISCAN* est basé sur un modèle maître/esclave et elle est implémentée sur la base du framework *BLADYG* décrit dans la Section 4.3. Dans ce framework, les esclaves (des machines de type worker) sont responsables de l'exécution d'un calcul spécifique sous le contrôle de la machine maître. De même, les données d'entrée doivent être divisées en sous-graphes. Dans ce qui suit, nous présentons les trois étapes principales de *DISCAN* : (1) le partitionnement de graphes, (2) le clustering distribué de graphes, et (3) la prise en compte des mises à jour.

4.4.1 Partitionnement de graphes distribués

Dans cette étape, *DISCAN* commence par la division du graphe d'entrée G en plusieurs partitions $P1, P2, \ldots P_n$ tout en maintenant la cohérence des données (structure globale du graphe). Pour garantir la propriété de cohérence lors de la division du graphe d'entrée, nous devons identifier une liste d'arêtes de coupe permettant d'obtenir une vue globale de G. Habituellement, le problème de partitionnement des graphes est classé dans la famille des problèmes NP-difficiles, qui doivent évaluer toutes les combinaisons pour obtenir le meilleur résultat de partitionnement. Pour cette raison, nous avons proposé un algorithme de partitionnement approximatif et distribué comme étant une étape préliminaire pour notre algorithme de clustering distribué. Dans l'étape de partitionnement, la machine maître divise de manière équitable le fichier du graphe d'entrée en sous-fichiers en fonction du nombre d'arêtes. Puis, elle envoie ces derniers à tous les workers. Chaque worker obtient une liste d'arêtes et de sommets de son sous-fichier. Il reçoit également la liste des sommets de workers voisins par la machine maître. Ceci est utile pour la détermination des sommets en frontière et donc pour l'identification des arêtes de coupe. Notons que ces dernières dénotent les arêtes ayant au moins un sommet en frontière.

4.4.2 Clustering distribué

Après avoir divisé le graphe d'entrée G en sous-graphes répartis entre les workers, *DISCAN* déclenche l'étape de clustering distribué (voir Algorithme 4). Cette étape est divisée à son tour en deux parties fondamentales : (1) le clustering local et (2) l'agrégation des résultats intermédiaires. Dans ce qui suit, nous décrivons en détail le principe de fonctionnement de ces dernières parties ainsi que le lien entre elles.

Étape 1 (Clustering local) : Le graphe d'entrée G est fragmenté en plusieurs sous-graphes/partitions (P), chacun d'eux est affecté à une machine esclave (ou *worker*). L'étape de partitionnement est effectuée conformément au nombre de machines esclaves. Pour éviter la perte d'informations au cours de l'étape de partitionnement (les arêtes reliant des sommets dans les différentes partitions), les sommets de frontière sont dupliqués dans les partitions voisines. Ensuite, pour chaque partition P_i, le clustering local est effectué sur les différentes partitions des machines esclaves. Nous considérons les cas où plusieurs conflits entre deux partitions P1 et P2

Algorithm 4 Distributed Clustering

Input : Graph G , parameters (μ, ϵ, α)
Output: $\mathbb{C}lusters, \mathbb{B}ridges, \mathbb{O}utliers$

`/* Divide G into subgraphs` $\mathbb{G} = \{G_1 G_2 .. G_\alpha\}$ `according to parameter` α `*/`
24 $\mathbb{P} \longleftarrow$ ***Partition*** *(G,α)*

`/* In parallel */`
25 **foreach** $P_i \in \mathbb{P}$ **do**
26 *Assign* P_i *to* W_i

`/* In parallel */`
`/* Step 1: Local clustering */`
27 **foreach** *Worker* $W_i \in \mathbb{W}$ **do**
28 *Let* P_i *the current partition*
Find the frontier vertices in P_i *and duplicate them into neighbor partitions*

`/* In parallel */`
29 **foreach** *Worker* $\in \mathbb{W}$ **do**
30 *Compute the structural similarity of a partition* P_i *using* V^f *list*
Retrive local Cores and Borders in P_i
Build local clusters in P_i
Find local Bridges and Outliers in P_i

`/* Step 2: Merging */`

All workers exchange theirs local clusters between them ; using Worker2Worker message
foreach *Worker* $W_i \in \mathbb{W}$ **do**
if *(*$C_1 \cap C_2 \cap .. \cap C_\alpha = \mathbb{V}$; *and* $\exists V_i \in \mathbb{C}ore$ **then**
$C \leftarrow$ ***Merge***$(C_1, C_2, ..C_\alpha)$
Send C *to the master*
else
Send local clusters to master
for V_i ***IN*** $\mathbb{O}utliers$ **do**
if *(*$V_i \in \mathbb{C}ore \cup \mathbb{B}order \cup \mathbb{B}ridges$ *)* **then**
Remove V_i *from the list of* $\mathbb{O}utliers$
for V_i ***IN*** $\mathbb{O}utliers$ **do**
$Nb_{Connections} \leftarrow 0$
for C_i ***IN*** $\mathbb{C}lusters$ **do**
if *(*$N(V_i) \cap C_i \neq \varnothing$*)* **then**
$Nb_{Connections} + +$
if *(* $Nb_{Connections} \geq 2$ *)* **then**
Add V_i *to* $\mathbb{B}ridges$
Remove V_i *from* $\mathbb{O}utliers$

Send Clusers,Bridges,Outliers to the master using Worker2Master message

ont eu lieu.

Étape 2 (Fusion) : En comparaison avec l'algorithme de référence *SCAN*, la distribution du calcul de similarité et l'étape de clustering local peuvent réduire le temps de réponse de *DISCAN*. Cependant, nous devons prendre en compte l'exactitude des résultats renvoyés. Pour garantir les mêmes résultats que *SCAN*, nous avons défini un ensemble de scénarios pour assurer l'étape de la fusion. Ces scénarios seront appliqués sur toutes les partitions de G . Ils sont répétés jusqu'à ce que toutes les partitions soient combinées. Pour chaque paire de partitions Pi et Pj, une fonction de fusion doit être exécutée pour combiner les résultats locaux de Pi et Pj. Ces résultats sont stockés dans des variables globales au niveau de la machine maître. L'Algorithme 4 présente plusieurs scénarios pour résoudre les éventuels conflits rencontrés.

Lemme 4.4.1. ***(Fusion de clusters locaux)*** *Soit $\mathbb{C}_1$ et $\mathbb{C}_2$ deux ensembles de clusters locaux dans différentes partitions P_1 et P_2, respectivement. $\exists c_1 \in \mathbb{C}_1$ and $\exists c_2 \in \mathbb{C}_2$, $Core(c_1) \cap Core(c_2) \neq \emptyset$.*

Démonstration. Soit C_i un cluster qui regroupe un ensemble de sommets de type bordure ou noyau. Si C_i partage au moins un sommet noyau c avec un autre cluster C_j , alors c a un ensemble de bordures dans C_i et C_j. De même, tous les sommets dans C_i et C_j peuvent être accessibles à partir de c. D'où, C_i et C_j devraient être fusionnés en un même cluster. □

Lemme 4.4.2. ***(De bruit à pont)*** *Soit $\mathbb{C}_1$ et $\mathbb{C}_2$ deux ensembles de clusters locaux dans les deux partitions P_1 et P_2, respectivement. $\exists$ c_1 et c_2 deux clusters qui appartiennent aux deux ensembles de clusters différents $\mathbb{C}_1$ et $\mathbb{C}_2$. De plus, $\exists$ o un sommet de type bruit dans les deux partitions P_1 et P_2 et $N(o) \cap c_1 \neq \emptyset$ et $N(o) \cap c_2 \neq \emptyset$.*

Démonstration. Si c_i et c_j $(i \neq j)$ partagent un sommet de type bruit o, cela signifie que o est faiblement connecté aux deux clusters c_i et c_j . Conformément à la Définition 1.3.7, o devrait être changé a un sommet de type pont. □

Lemme 4.4.3. ***(De Pont à bruit)*** *Soit c_1 et c_2 deux clusters locaux respectivement dans les deux partitions P_1 et P_2. $\exists$ un pont b en fonction uniquement de deux clusters c_1 et c_2. Lorsque c_1 et c_2 seront fusionnés en un seul cluster (pendant l'étape de fusion), b devrait être changé à un sommet de type bruit.*

Démonstration. Soit c_i et c_j deux clusters qui ont un ensemble de sommets (bordures ou noyaux) et un ensemble de ponts avec d'autres clusters locaux et $\exists$ b un sommet de pont uniquement en fonction des deux clusters c_i et c_j où $i \neq j$. Dans l'étape de fusion et selon le Lemme 4.4.1, si un ou plusieurs clusters partagent au moins un sommet noyau, nous les fusionnons en un même cluster. Dans ce cas $(c_i,c_j) \Rightarrow C$ ce qui rend b faiblement connecté à un seul cluster C. Ainsi, selon la Définition 1.3.8, alors b devrait changer son statut de pont à bruit. □

4.4.3 Mise à jour du clustering

Considérons un grand graphe dynamique G, qui subira plusieurs modifications au fil du temps, telles que l'ajout ou la suppression d'arêtes et/ou de sommets. Lorsqu'un clustering est effectué, les approches classiques relancent le clustering à partir de zéro, ce qui est évidemment très coûteux surtout dans les grands graphes. Dans ce travail, nous proposons un algorithme incrémental et distribué pour le clustering de grands graphes dynamiques. L'idée principale de notre approche est de déterminer à chaque mise à jour les sommets et les arêtes concernés par cette mise à jour, et en fonction de cela, nous appliquons les nouvelles mises à jour sur les anciens résultats de clustering.

Supposons que $G = (V, E)$ va subir plusieurs mises à jour U au fil du temps. U peut ajouter/supprimer des sommets ou des arêtes de/vers G. Ces modifications peuvent affecter les valeurs de similarité et certains statuts de nœuds du graphe (bordure, noyau, etc.). Par conséquent, le schéma de clustering peut être affecté dans plusieurs situations.

Définition 4.4.1. *Chaque sommet $u_i \in U$ génère des modifications de plusieurs similarités structurelles, qui sont définies comme les arêtes affectées E_A. Ces sommets changent également le statut de plusieurs sommets et sont notés comme sommets affectés V_A.*

Définition 4.4.2. *(Arêtes affectées) Lors de l'ajout d'un sommet v_{new} lié à un sommet existant v_{old}, une nouvelle arête (v_{new}, V_{old}) sera créée et un ensemble d'arêtes noté E_A sera marqué comme arêtes affectées et doit être mis à jour.*

Notez que $e = (u,v) \in E_A$, où $u \in V_{old}$ et $v \in N(V_{old})$. Dans le cas de la suppression d'un sommet V_{TBD}, un ensemble d'arêtes noté E_{TBD} doit être supprimé de G. Ainsi, un V_c de sommets est concerné par cette mise à jour. Ici $V_c = (u, v) \in E_{TBD} \setminus V_{TBD}$ et, d'après V_c, les

arêtes affectées sont $E_A = E_i \subset G$ *et* $E_i = (u, v)$ / $\{u,v\} \in V_c$.

Lors de l'ajout ou de la suppression d'une arête (v_i, v_j)*, la liste des voisins des deux sommets* v_i *et* v_j *sera modifiée, ce qui peut produire plusieurs autres arêtes. Les arêtes affectées mettent à jour leurs similitudes structurelles.* E_A *est un ensemble d'arêtes* $(u, v) \in E$ *et* $v_i = u$ *ou* $v_j = v$.

Définition 4.4.3. *(nœuds affectés) Pour chaque mise à jour sur G, certaines similarités structurelles peuvent être modifiées. Ainsi, certains sommets changent de statut (bordure, sommets noyaux). Ces sommets sont définis comme des sommets affectés* V_A*.* V_A *peut être divisé en deux sous-ensembles : (i) directement affectés* V_{AD}*, qui mettent à jour leur statut en fonction directement de* E_A*, (ii) indirectement affectés* V_{ID} *qui dépendent du statut de* V_{AD}*. Par exemple, lorsqu'un sommet noyau change son statut en valeur aberrante ou bordure, tous ces sommets bordures seront affectés indirectement en fonction de cette mise à jour. Soit* E_A=(v_1, v_2) *suivant une mise à jour* V_{AD} =$V_{AD} \cup v_1 \cup v_2$*, tandis que* V_{AI} *sont tous voisins de* V_{AD}*. D'autres sommets peuvent être affectés indirectement par plusieurs mises à jour et dépendent des clusters modifiés. Ces sommets sont définis comme les ponts affectés et notés* V_{AB}*.* V_{AB} *est la liste des valeurs aberrantes ou des ponts qui sont connectés aux clusters affectés décrits dans la Définition 4.4.4.*

Définition 4.4.4. *(Clusters affectés) Les sommets affectés dans plusieurs situations peuvent affecter le schéma de clustering. Les clusters affectés (concernés) représentent le sous-ensemble de clusters qui contiennent un ou plusieurs sommets affectés, désignés par* $C_{affected}$*.*
C_{affect}=$c \in \mathbb{C}$ *tel que* $v \in (v_{AD} \cup v_{ID})$ *et* $v \in c$.

Lemme 4.4.4. ***Changement de similarité structurelle des arêtes affectées***

Les arêtes affectées E_A *dépendent principalement de la Définition 1.3.1. La valeur de similarité est basée sur les voisins des deux sommets d'une arête. Ainsi, chaque mise à jour (ajout/suppression d'une arête/sommet) peut modifier la liste des voisins de certains sommets. En conséquence, certaines arêtes doivent changer leur similarité structurelle.*

Lemme 4.4.5. ***Modification de l'état du sommet affecté***

Selon les définitions 1.3.4 et 1.3.3, pour certaines mises à jour dans la similarité structurelle, lorsque la valeur de similarité dépasse ou diminue le seuil ϵ *le statut de cette arête peut être modifié à une connexion forte ou faible. Ainsi, ce changement peut affecter le statut de chaque*

sommet $v \in V_{AD}$. Par conséquent, selon la définition 1.3.5, le V_{DA} peut modifier indirectement le statut d'autres sommets qui sont définis comme des sommets indirectement affectés V_{IA}. Rationnellement, ces changements peuvent affecter le mappage des clusters, et plusieurs clusters peuvent être modifiés. Ainsi, selon la définition 1.3.7, un sommet $v \in \mathbb{B}$ où v ne dépend que de deux clusters c_1 et c_2. Lorsque c_1 et c_2 seront fusionnés en un cluster c_{new}, v sera faiblement connecté avec seulement c_{new}. Dans ce cas et selon la définition 1.3.7, v sera un sommet aberrant. De la même manière, lorsque nous avons un cluster c et que v est une valeur aberrante à c avec deux connexions faibles liées à v_1 et v_2, après quelques mises à jour, c est divisé en deux clusters c_1 et c_2 lorsque v_1 et v_2 sont respectivement dans c_1 et c_2. Dans ce cas, v devient un pont entre les nouveaux clusters c_1 et c_2 selon la connexion de v avec v_1 et v_2.

Algorithm 5 Algorithme *DISCAN*

Input : Initial graph G as a text file, parameter (NP number of partitions), a new update U
Output: $\mathbb{C}, \mathbb{B}, \mathbb{O}$

```
/* Global affected vertices and clusters in each worker                    */
```
31 $\mathbb{G}lobalAffectedVertices$=$\emptyset$ $\mathbb{G}lobalAffectedClusters$=$\emptyset$ `/* Step 1: Graph maintenance */`
```
/* Update the inital graph in each new update and get the affected vertices and
   the affected clusters                                                   */
```
32 *Master machine : send a new update u to all workers* `/* In parallel */`
33 **foreach** $Worker_i$ $w_i \in \mathbb{W}$ **do**
34 *Update the current partition according to U Get E_A Recompute the similarities of E_A Get the immediately affected vertices V_I Share the immediately affected vertices V_I with all workers Get the indirectly affected vertices V_{Ind}, and the affected clusters C_A*
```
/* Step 2: Local clustering schema maintenance                             */
/* Update the old local clustering schema according to the global affected
   vertices                                                                */
```
35 **foreach** $Worker_i$ $w_i \in \mathbb{W}$ **do**
36 *Check Core vertices from GlobalAffectedVertices Check Border vertices from GlobalAffectedVertices Check affected clusters Check affected bridges*
```
/* Step 3: Merge the new updates                                           */
```
37 **foreach** $Worker_i$ $w_i \in \mathbb{W}$ **do**
38 *Merge all affected clusters from all workers Check new Bridge vertices according to the new clusters Check the remaining outlier vertices*
```
/* Reset the global affected vertices and clusters in each worker          */
```
39 $\mathbb{G}lobalAffectedVertices$=$\emptyset$ $\mathbb{G}lobalAffectedClusters$=$\emptyset$
40 *Send $\mathbb{C}$,$\mathbb{B}$,$\mathbb{O}$ to master using Worker2Master message*

L'algorithme 5 décrit les principales étapes de *DISCAN*. La méthode *DISCAN* fournit une maintenance du graphe en temps réel et un clustering par micro-lots (ou *micro-batch*). De

plus, *DISCAN* effectue le nouveau clustering après plusieurs modifications, afin d'optimiser le clustering incrémental. Nous vous présentons, ci-dessous, les principales étapes de *DISCAN* :

Étape 1 : Maintenance du graphe. Dans cette étape, *DISCAN* exécute des opérations de maintenance sur le graphe en temps réel. A chaque mise à jour $u \in U$, il (1) met à jour la structure du graphe G, (2) vérifie les arêtes affectées E_A, et (3) recalcule les similarités de E_A (lignes 4- 12 de l'algorithme 5). Ensuite, il récupère V_A en fonction de E_A afin de les mémoriser dans une variable globale. Comme discuté dans la définition 4.4.4, V_A peut affecter plusieurs clusters dans la partition actuelle ou dans d'autres partitions. Ainsi, chaque worker partage son V_A avec tous les workers afin que chacun détermine les clusters concernés et les sauvegarde. Une fois les clusters affectés fixés, ils affectent certains ponts ou sommets aberrants. Les ponts concernés doivent donc être ajoutés à la liste des sommets concernés à vérifier à l'étape suivante.

Étape 2 : Clustering local incrémental. Cette étape est effectuée en mode de traitement par micro-batch, après un certain nombre de mises à jour ou en utilisant une fenêtre temporelle (ou un intervalle de temps). Dans chaque lot/batch, *DISCAN* vérifie d'abord les sommets principaux. La vérification est effectuée uniquement sur les sommets affectés V_A, où en général les $|V_A| \ll |V|$. Ensuite, *DISCAN* vérifie les sommets de bordure comme dans le premier clustering local (Algorithme 4) mais en utilisant seulement E_A, puisque les sommets de bordure dépendent des connexions fortes avec les sommets du noyau. Pour cette raison, *DISCAN* vérifie les sommets principaux selon E_A uniquement. Après cela, les noyaux et les bordures mis à jour modifieront le schéma de clustering en fonction des sommets affectés (sommets noyaux, bordures) comme décrit dans la définition 4.4.4. Il y a cinq cas possibles en raison de chaque mise à jour d'un graphe : (1) diviser un cluster en petits clusters, (2) supprimer un cluster existant, (3) créer de nouveaux clusters, (4) mettre à jour les clusters existants et (5) fusionner deux clusters ou plus.

Considérons un ensemble de noyaux et leurs bordures, et les noyaux ont de fortes connexions entre eux. Après la suppression d'une connexion forte entre deux cœurs c_1 et c_2, les clusters doivent être divisés en sous-clusters en fonction de c_1 et c_2. Pour le cas (2), chaque cluster est construit sur une liste de nœuds noyaux et leurs bordures. Si un cluster a un sommet noyau, il doit être supprimé. Parfois, une nouvelle mise à jour crée un sommet aberrant vers un sommet

principal c_{new}. Si ce sommet a une connexion forte avec tout autre ancien noyau c, il rejoint le cluster de c, comme dans le cas (4). Si c_{new} a des connexions fortes avec un autre noyau, il construit un nouveau cluster, comme dans le cas (3). Dans le dernier cas (5), s'il existe une nouvelle connexion forte entre deux sommets noyaux dans deux clusters différents, alors on fusionne ces derniers.

Afin de traiter tous les cas mentionnés, nous procédons à la suppression de tous les clusters affectés, puis nous effectuons un nouveau clustering de la même manière que celui présenté dans la définition 1.3.6, en utilisant uniquement les sommets affectés (c'est-à-dire les sommets noyaux et les bordures). Ce nouveau clustering est réalisé uniquement sur les sommets modifiés. *DISCAN* utilise ensuite les nouveaux clusters ainsi que les sommets affectés restants pour vérifier s'ils représentent de nouveaux sommets de bordure.

Étape 3 : Fusionner les nouvelles mises à jour. Après chaque lot/batch, chaque worker commence à fusionner les nouvelles mises à jour (voir Algorithme 5 lignes 19-23). La machine maître obtient tous les sommets affectés de tous les workers afin de faciliter la combinaison des nouveaux changements. Ici, certains clusters doivent être vérifiés. Dans chaque lot, le maître ne conserve que les clusters passés inchangés et demande à tous les workers d'obtenir les clusters modifiés. Ainsi, tous les workers combinent les clusters mis à jour, y compris éventuellement de nouveaux clusters. Dans l'étape de fusion, *DISCAN* utilise les mêmes scénarios que dans le premier clustering (voir Lemme 4.4.1). Si au plus un cluster partage au moins un sommet noyau, ils peuvent être fusionnés en un seul cluster. À l'étape suivante, après avoir obtenu les nouveaux clusters, la machine maître demande à tous les workers d'obtenir leurs sommets de type bordure. Ensuite, chaque worker filtre uniquement les sommets affectés qui appartiendront à ses ponts locaux, et les envoie au maître. Le reste des sommets affectés sera ajouté à la liste globale des valeurs aberrantes. Enfin, *DISCAN* initialise les sommets et les clusters affectés globalement dans des listes vides qui seront utilisés dans le prochain lot.

4.5 Résultats

4.5.1 Temps de réponse du clustering distribué initial

DISCAN est conçu pour traiter les graphes dynamiques. Cependant, il commence par effectuer un clustering initial du graphe d'entrée. Ensuite, il traite des changements dans le graphe. Nous avons évalué l'accélération de la première étape de *DISCAN* et nous l'avons comparée avec le *SCAN* de base et ses variantes [Inoubli et al., 2022].

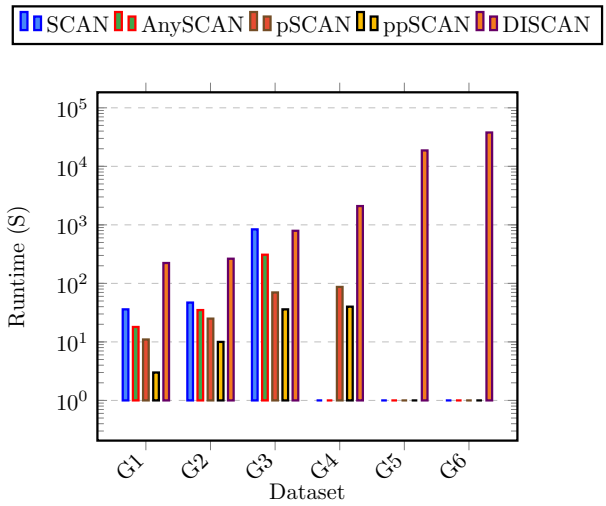


FIGURE 4.2 – Impact de la taille du graphe sur le temps de traitement des variantes *SCAN* et *DISCAN*. [Inoubli et al., 2022]

Comme le montre la Figure 4.2, notre approche est plus lente que les autres algorithmes, dans le cas de petits graphes ($G1$, $G2$ et $G3$). De plus, il y a un très grand écart entre *DISCAN* et les autres algorithmes, en particulier avec *pSCAN* et *ppSCAN*. Cet écart se réduit lorsque la taille du graphe augmente (cas du jeu de données $G4$). Le diagramme à bâtons de la Figure 4.2, montre un écart de 12x entre *DISCAN* et *SCAN* de base avec le jeu de données $G1$ et 2x uniquement avec le jeu de données $G4$. Nous remarquons que l'écart entre *DISCAN* et *ppSCAN* dépend principalement de la taille du jeu de données utilisé. Par exemple, avec le jeu de données $G1$, l'écart entre *DISCAN* et *ppSCAN* atteint 20x, alors qu'il est réduit à 11x avec le jeu de données $G4$. Cela peut s'expliquer par l'étape d'élagage de *pSCAN*, qui dispense plusieurs calculs de similarité lors de l'étape de clustering. Il est important de mentionner que *DISCAN* est une implémentation distribuée de *SCAN*, ce qui n'est pas le cas des autres algorithmes

étudiés, car ils sont centralisés. Cela entraîne des coûts supplémentaires liés à la distribution des données, à la synchronisation et à la communication. La Figure 4.2 montre également qu'avec des configurations matérielles modestes, seul *DISCAN* peut passer à l'échelle avec de grands graphes comme $G5$ et $G6$.

4.5.2 Temps de réponse et clustering incrémental

Dans un premier temps, nous avons commencé par exécuter tous les algorithmes sur les graphes utilisés. Ensuite, nous avons périodiquement ajouté de nouveaux lots de mises à jour de différentes tailles. La Figure 4.3 montre le temps d'exécution des algorithmes testés avec différents graphes. Dans tous les graphes utilisés, *DISCAN* est surpassé par les autres algorithmes, à l'exception de *pSCAN* qui ne supporte pas le graphe $G5$. Malgré que *pSCAN* et *ppSCAN* ré-exécutent le clustering à partir de zéro, ils sont plus rapides que *DISCAN* dans le cas de $G2$, et ont un temps proche dans le cas de $G3$. Dans le cas de $G4$, *DISCAN* présente initialement un meilleur comportement que *pSCAN* et *ppSCAN*. En fait, *ppSCAN* et *pSCAN* sont plus rapides que *DISCAN*, mais l'écart de temps d'exécution commence à se réduire entre $G2$ et $G4$. Cependant, lorsque nous ajoutons un nouveau lot aux graphes initiaux, *DISCAN* devient plus rapide que *pSCAN* et *ppSCAN*. Dans le cas de $G3$, la Figure 4.3 montre un écart de 2x entre *DISCAN* et *pSCAN* et un temps d'exécution presque proche de *DISCAN* et *ppSCAN*. De plus, l'écart entre *DISCAN* et *pSCAN* commence à augmenter, pour finalement atteindre 3x, et 10% entre *DISCAN* et *ppSCAN*.

4.5.3 Scalabilité de *DISCAN*

La Figure 4.4 montre la scalabilité de *DISCAN* par rapport au nombre de nœuds de calcul, avec les ensembles de données $G2$ et $G3$, et pour différents lots de mise à jour en utilisant les paramètres par défaut (ϵ=0,5 et μ=3).

Dans l'ensemble, le nombre de workers affecte le temps d'exécution en fonction de la taille de l'ensemble de données utilisé et du nombre de mises à jour. Dans la Figure 4.4 avec le jeu de données $G2$, toutes les courbes ont presque la même forme. Le temps de réponse est faible avec 2 à 4 workers, mais ce n'est pas toujours valable, car cette amélioration dépend du nombre de

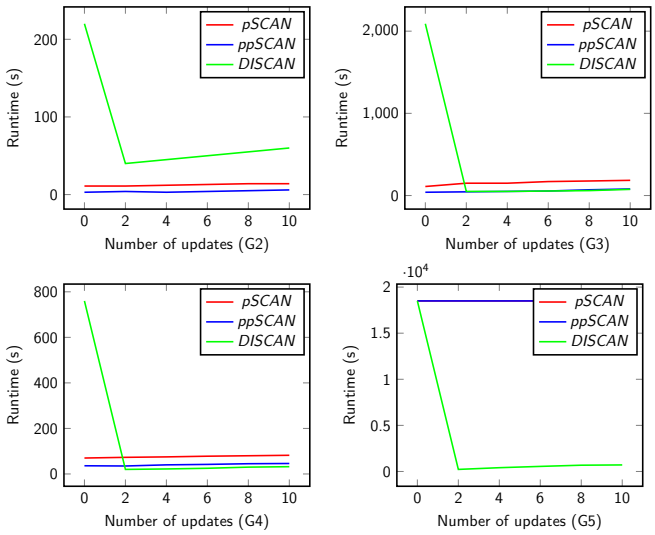


FIGURE 4.3 – Impact du nombre de mises à jour sur le temps de traitement de *pSCAN*, *ppSCAN* et *DISCAN* (ϵ=0.5, μ=3). [Inoubli et al., 2022]

mises à jour. Les améliorations sont comprises entre 10% et 50%, respectivement, pour les mises à jour 2000 et 10000. Ensuite, les courbes croissent jusqu'à 8 workers, puis redescendent. Dans le cas du jeu de données *G3*, la scalabilité devient très élevée. Le temps de réponse diminue en fonction du nombre de workers. Cette amélioration est d'environ 30% et 40% selon le nombre de mises à jour. Ce comportement peut s'expliquer par le nombre de sommets et de clusters affectés. Lorsque ce nombre est petit, les ressources nécessaires (par exemple, les workers) sont petites. Néanmoins, avec de nombreux workers, le temps de traitement de *DISCAN* devient élevé en raison du coût de communication.

Des expérimentations supplémentaires de la méthode *DISCAN* (exactitude de la méthode, impact du partitionnement, impact de la taille du lot et des types d'arêtes de mise à jour sur les performances de la méthode) sont accessibles dans [Inoubli et al., 2022].

4.6 Discussion et conclusions

Dans ce chapitre, nous avons présenté nos approches distribuées d'analyse de grands graphes complexes et dynamiques. Tout d'abord, nous avons présenté BLADYG, une approche générique pour le calcul distribué sur des grands graphes dynamiques. BLADYG a montré son efficacité et son applicabilité pour la résolution de plusieurs problèmes comme le clustering distribué et

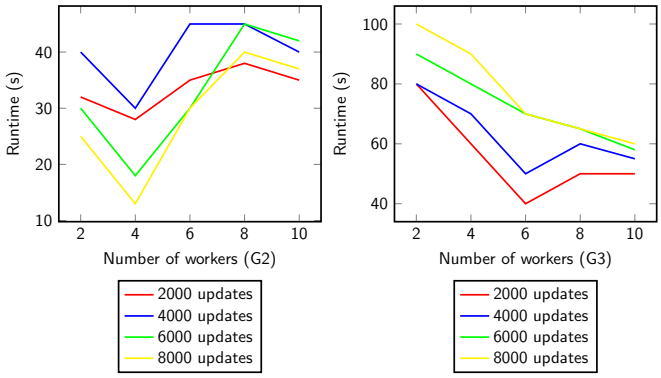


FIGURE 4.4 – Impact du nombre de workers sur le temps de traitement de *DISCAN* (ϵ=0.5, μ=3). [Inoubli et al., 2022]

incrémental des grands graphes (voir Section 4.4), la recherche des régions les plus cohérentes dans des graphes dynamiques [Aridhi et al., 2016a] et le partitionnement de graphes dynamiques à grande échelle [Sakouhi et al., 2016]. Cependant, BLADYG présente certaines limitations comme la quantité importante de données/messages échangés entre les machines distribuées ou aussi sa dépendance aux méthodes de partitionnement de graphes existantes.

Nous avons présenté aussi dans ce chapitre la méthode *DISCAN* pour le clustering distribué et incrémental des grands graphes. DISCAN a été évaluée et comparée avec des méthodes de clustering à grande échelle et a montré son efficacité. Cependant, DISCAN présente quelques limitations comme la méthode de partitionnement utilisée qui peut engendrer un nombre très élevé de nœuds en frontière et donc un temps de traitement plus important. Une autre piste d'amélioration de DISCAN consiste à utiliser une mémoire partagée à la place d'une communication par transmission de messages entre les workers dans le but de diminuer la consommation de ressources de bande passante et par conséquent diminuer le temps de calcul.

Chapitre 5

Apprentissage de plongements dans les grands graphes de connaissances distribués

5.1 Introduction

En raison de la taille importante des graphes de connaissances (KG) du monde réel, une seule unité de calcul n'est pas suffisante pour stocker un KG et apprendre les plongements en un temps raisonnable. Dans ce contexte, il serait intéressant d'implémenter des méthodes de plongement de graphes de connaissances (ou *Knowledge Graph Embedding (KGE)*) dans un environnement distribué où un KG est distribué sur un ensemble de machines et les plongements seront appris sur les machines en parallèle. À ce jour, quelques frameworks distribués existent dans la littérature pour l'apprentissage de plongements géométriques, tels que GraphVite [Zhu et al., 2019], Pytorch-BigGraph (PBG) [Lerer et al., 2019] et DGL-KE [Zheng et al., 2020]. GraphVite et PBG utilisent l'API PyTorch-Distributed pour mettre en oeuvre un réseau de communication et d'échange de données. Cependant, l'utilisation des APIs de bas niveau est une tâche difficile pour le développement et le déploiement [Sheikh et al., 2022a]. Récemment, Moritz *et al.* [Moritz et al., 2018a] ont développé la plateforme Ray pour faciliter le développement et le déploiement d'applications distribuées de manière simple en masquant les détails de bas niveau du développement et du

déploiement.

Dans le cadre de la thèse de Kamrul Islam [Islam, 2022], nous avons proposé un framework distribué basé sur la plateforme Ray pour les approches géométriques de KGE. Le framework proposé utilise le stockage distribué, l'apprentissage distribué et les mécanismes de partage de gradient Ring-Allreduce. Dans l'ensemble, le nœud principal (maître) crée plusieurs partitions d'un KG et envoie chaque partition à chaque worker. Chaque worker génère ensuite des exemples négatifs en utilisant notre méthode d'échantillonnage négatif simple SNS [Islam et al., 2022] et calcule le gradient pour un lot d'exemples positifs et négatifs. Chaque worker partage son gradient avec d'autres workers par le mécanisme Ring-AllReduce et les gradients sont ensuite agrégés dans chaque worker pour mettre à jour ses plongements. Comme tous les workers contiennent des plongements d'entités et de relations, le modèle formé est évalué sur un worker sélectionné au hasard. Notre framework distribué proposé est basé sur une architecture All-Reduce où toutes les machines fonctionnent (pour l'apprentissage des plongements) en tant que workers contrairement à l'architecture maître-esclave où le maître gère les ressources et seuls les workers apprennent les plongements.

Dans ce chapitre, je présente principalement le framework proposé pour l'apprentissage de plongements dans des graphes de connaissances distribués. Je détaillerai en particulier le modèle distribué d'apprentissage des plongements.

5.2 Une approche distribuée de KGE

L'architecture de notre framework est illustrée dans la Figure 5.1.

Notre framework se compose de deux composants : un nœud/machine principale et quelques machines workers. Le nœud principal est un nœud spécial qui a quelques tâches spéciales comme un maître dans une architecture maître-esclave. Le nœud principal est chargé de récupérer le KG et de le partitionner en petites partitions. Le nœud principal initie également les communications entre tous les nœuds de calcul, envoie une partition à chaque nœud de calcul et conserve une partition pour lui-même. À la fin du processus d'apprentissage, le nœud principal est également chargé de renvoyer les plongements appris. Pendant la phase d'apprentissage, le nœud principal agit également comme un nœud de travail (ou *worker*). Chaque worker reçoit une partition du

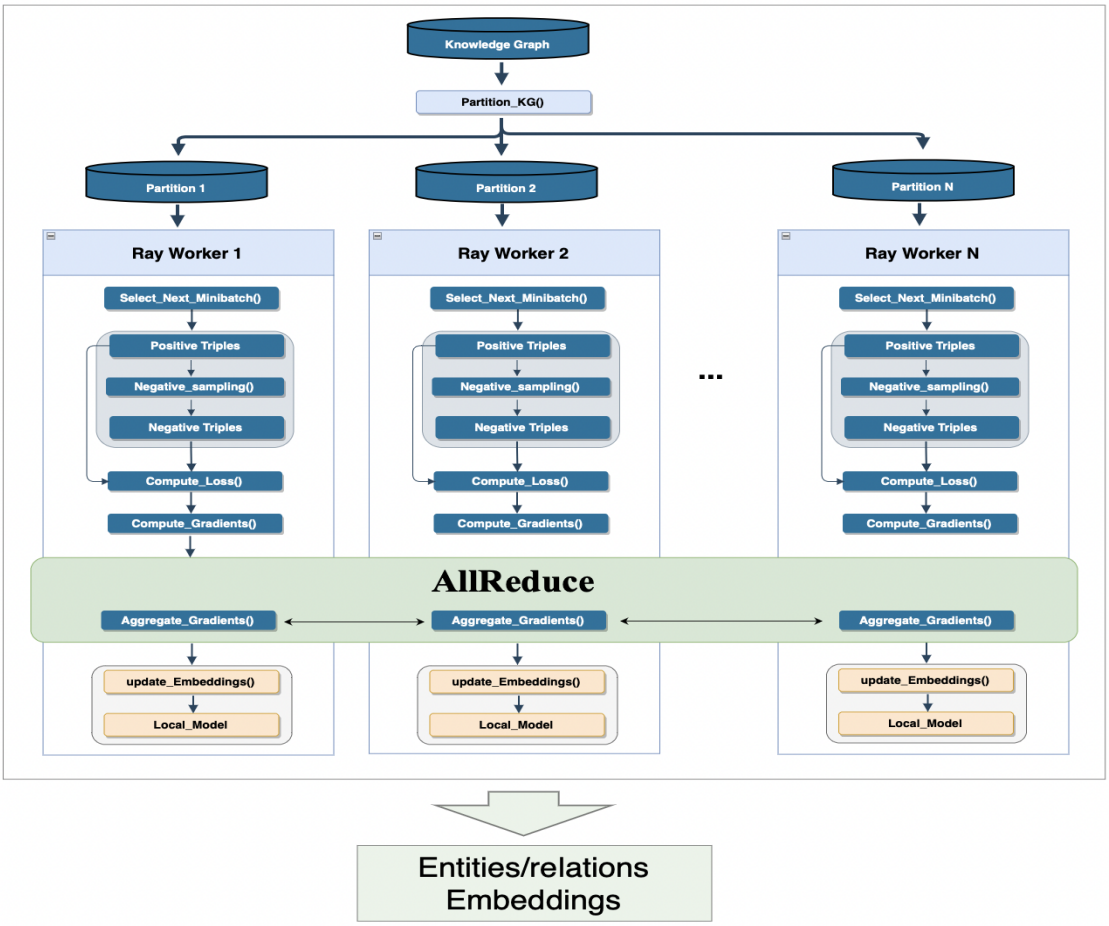


FIGURE 5.1 – Architecture du framework distribué pour l'apprentissage de plongements. [Islam, 2022]

KG et est responsable du calcul des gradients et de l'apprentissage des plongements des entités et relations du KG. Chaque worker contient un sous-composant AllReduce pour recevoir les gradients des autres workers et pour calculer ses gradients finaux.

Décrivons maintenant plus en détail les principales étapes du système distribué à savoir : (1) l'établissement d'un réseau de communication, (2) le partitionnement du KG, et (3) l'entrainement du modèle de plongement.

5.2.1 Établissement d'un réseau de communication

Notre framework commence par établir un réseau de communication entre toutes les machines/workers dans un environnement distribué. Soit N le nombre total de machines. Deux

architectures populaires de réseaux de communication sont examinées :

1. Architecture du serveur de paramètres partagés : dans cette architecture, $N-1$ machines (workers) sont utilisées pour l'apprentissage sur $N-1$ partitions de données et une machine est utilisée comme serveur de paramètres. Les données sur le serveur de paramètres sont partagées entre tous les workers. La disposition des machines dans le réseau pourrait être imaginée comme une topologie en étoile où la machine serveur de paramètres reste au centre et tous les workers sont aux extrémités. Les nœuds de calcul utilisent leurs partitions de données pour calculer les pertes et les gradients, et envoient leurs gradients au serveur de paramètres pour stocker et mettre à jour les paramètres des modèles d'apprentissage automatique [Li et al., 2014].
2. Architecture AllReduce : dans cette architecture, chaque machine fonctionne comme un nœud de calcul et stocke tous les paramètres pouvant être entraînés [Bruck and Ho, 1993, Thakur et al., 2005]. Tous les workers calculent les pertes et les gradients en fonction de leurs partitions de données et partagent leurs gradients avec tous les workers via un composant spécial appelé AllReduce dans chaque worker. Tous les workers agrègent tous les gradients et ils ont les mêmes gradients. Contrairement au serveur de paramètres partagés, AllReduce a N workers pour N machines dans les systèmes distribués.

L'avantage d'une meilleure utilisation des ressources de l'architecture AllReduce par rapport à l'architecture du serveur de paramètres nous motive à concevoir le réseau de communication basé sur une architecture AllReduce. Pour le cas AllReduce, deux architectures bien connues sont Tree-AllReduce et Ring-AllReduce. Dans l'architecture Tree-AllReduce, chaque worker communique avec ses workers gauche et droit [Yang and Amazon, 2018]. Dans l'architecture Ring-AllReduce, les nœuds workers sont organisés en anneau où chaque worker communique avec son nœud précédent [Sergeev and Del Balso, 2018].

5.2.2 Partitionnement du KG

Le processus d'apprentissage commence avec le nœud principal récupérant les triplets du graphe d'entrée. Bien qu'il existe de nombreux algorithmes pour partitionner des graphes simples, MCS [Zhong et al., 2018] est le seul algorithme de partitionnement de graphes de connaissances

dans la littérature à notre connaissance. Comme le code source de cet algorithme n'est pas accessible au public, nous implémentons un algorithme de partitionnement aléatoire de graphes. Dans le partitionnement aléatoire, nous divisons aléatoirement les triplets du KG en un nombre de partitions où chaque sous-graphe a un nombre presque égal de triplets. Le nœud principal produit N partitions d'un KG pour N nœuds ($N-1$ workers et 1 nœud principal) dans notre environnement distribué.

Algorithm 6 The head node

Input: A knowledge graph (KG), number of workers (N), embedding method (KGE), Random partitioner (KG_Splitter), total number of epochs (T), number of batches (B), embedding optimizer (Optimizer), simple negative sampling (SNS)

```
/* Partition the KG using a KG partitioning method.                    */
```
41 $Partitions[1:N]$ ← KG_Splitter(KG,N) Workers ← ∅ `/* Create a temporary KGE model */`

42 model_tmp ← KGE(KG) model_tmp.embeddings ← initialize_embeddings() **foreach** $Part_{KG} \in Partitions[2:N]$ **do**

```
    /* Initialize a worker                                              */
```
43 worker ← ∅ worker.*triples* ← $Part_{KG}$.triples worker.*model* ← model_tmp worker.*negative_sampler* ← SNS Workers.insert(worker)

```
/* Transform the head node to a normal worker                          */
```
44 worker ← head_node worker.*triples* ← $Partitions$[1].triples worker.*model* ← model_tmp worker.*negative_sampler* ← SNS Workers.insert(worker) `/* Send signal to Ray platform to start training of all workers */`

45 Signal_parallel_training(Workers)

5.2.3 Entraînement du modèle de plongement

Après le partitionnement du graphe, le nœud principal instancie un modèle de plongement en initialisant les plongements d'entités et de relations de manière aléatoire à partir de distributions uniformes/gaussiennes [Wang et al., 2017]. Notre framework permet d'entraîner des modèles géométriques de KGE tels que TransE [Bordes et al., 2013], TransH [Wang et al., 2014], *TransD* [Ji et al., 2015], *DistMult* [Yang et al., 2015], *ComplexX* [Trouillon et al., 2016] et *RotationE* [Sun et al., 2018]. Le nœud principal envoie des copies de ce modèle à d'autres $N-1$ workers. Le nœud principal envoie un signal au backend Ray pour démarrer l'apprentissage parallèle. En même temps, il se transforme en worker normal. Les étapes de travail dans le nœud principal sont écrites dans l'Algorithme 6. Par conséquent, tous les N workers de notre framework ont les mêmes plongements pour toutes les entités et relations.

Chaque worker suit le même processus d'apprentissage de plongement qui est similaire dans une certaine mesure à un modèle de plongement centralisé. Chaque worker traite ses triplets en mode batch, *c'est-à-dire*, il crée un nombre fixe de lots de triplets. Chaque worker suit ensuite les étapes d'apprentissage suivantes pendant T fois (époques) pour mettre à jour son modèle de plongement.

Calcul du gradient

La machine worker récupère un lot de triplets positifs S_m de taille m à partir de sa partition et génère un lot correspondant de triplets négatifs S'_m en utilisant une méthode d'échantillonnage de triplets négatifs. Nous utilisons notre méthode SNS pour générer des triplets négatifs de haute qualité où seul l'ensemble d'entités de la partition actuelle est utilisé pour générer l'ensemble candidat. Les lots de triplets positifs et négatifs sont ensuite utilisés pour former le modèle de plongement. Nous considérons la stratégie d'apprentissage par paires où l'objectif est d'optimiser les plongements d'entités et de relations en minimisant la perte totale par paires (L) pour le lot tel que défini dans la Section 3.2, équations 3.1 et 3.2.

Agrégation des gradients et mise à jour des plongements

Après avoir calculé les gradients, le composant AllReduce de chaque worker partage les gradients de son modèle avec les autres workers. Après le partage des gradients, chaque worker aura des copies des gradients de tous les autres workers. Le worker agrège ces gradients en les additionnant simplement. La Figure 5.2 illustre un exemple de partage de gradients entre trois workers où chaque worker contient trois gradients.

Chaque machine worker met à jour son modèle en fonction de ses gradients agrégés. Notez que les modèles appris dans tous les workers ont les mêmes plongements car tous les workers utilisent les mêmes gradients pour les mettre à jour.

Chaque machine worker répète l'étape d'apprentissage de plongements décrite ci-dessus pour tous les lots et pour toutes les époques. L'algorithme 7 résume toutes les étapes d'une machine worker. Comme tous les workers ont le même modèle appris, notre framework sélectionne le modèle de plongement du nœud principal comme modèle de sortie. Le modèle appris est utilisé dans la tâche de prédiction de liens.

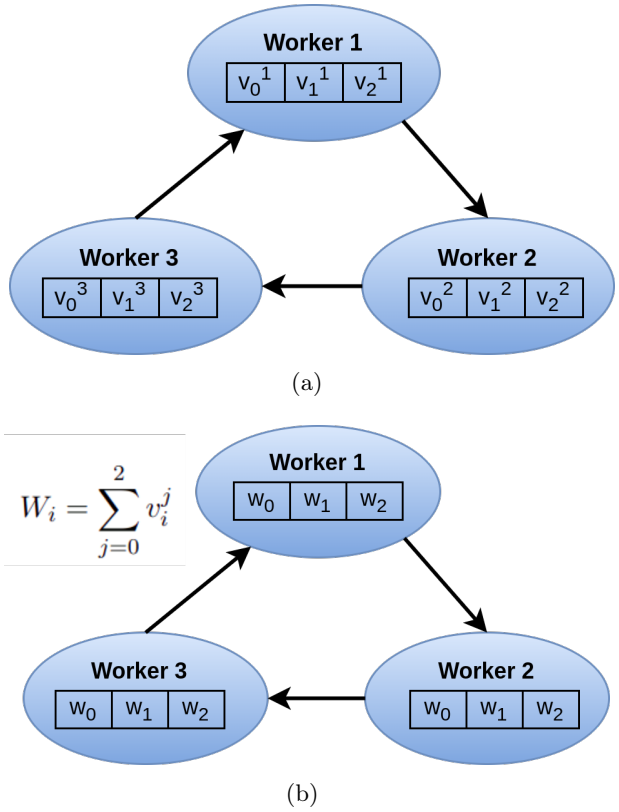


FIGURE 5.2 – Illustration du mécanisme d'agrégation de gradient Ring-AllReduce pour une entité/relation spécifique dans un espace latent tridimensionnel. (a) Chaque worker i calcule le vecteur gradient v_0^i, v_1^i, v_2^i pour l'entité/relation. (b) Chaque worker partage ses gradients avec d'autres workers et agrège tous les gradients. [Islam, 2022]

5.3 Résultats

Pour tester notre framework distribué, nous avons utilisé Grid 5000 [Balouek et al., 2013], un ensemble de machines de calcul haute performance pour le développement d'applications distribuées à grande échelle. Toutes les expériences sont réalisées sur des clusters avec des GPU Nvidia GeForce RTX 2080 Ti (11 Go). Pour les expériences centralisées, nous n'utilisons qu'un seul GPU sans le framework Ray.

Pour comparer notre framework distribué avec une solution centralisée, nous exécutons la solution centralisée pour 500 époques pour tous les ensembles de données de test. Dans notre framework distribué, nous choisissons 2,4,6,8,10 comme nombre de workers. Nous avons défini un nombre d'époques égale à 500, un nombre de lots égale à 120 pour les jeux de données FB15K et à 250 pour les jeux de données YAGO3-10. En ce qui concerne l'optimiseur, nous utilisons

Algorithm 7 Worker computation

Input: A worker (worker), number of epochs (T), number of batches (B), an embedding optimizer (Optimizer)

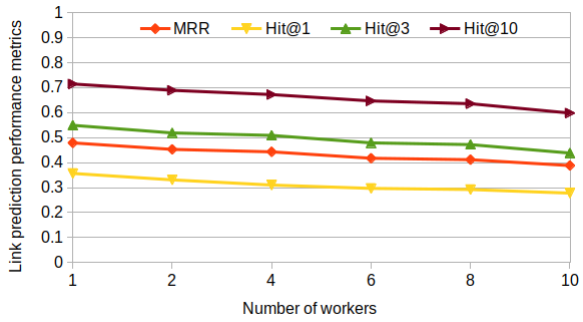
Output: Trained embedding model ($\text{KGE}_{trained}$)

```
/* Initialize the SNS method using information for the local partition.  */
worker.SNS.initialize()  foreach epoch ∈ {1, 2, 3, ....., T} do
    foreach batch ∈ {1, 2, 3, ....., B} do
        gradient ← ∅  /* Sample a batch of positive and negative triples.  */
        Triples.positive ← Extract_samples(worker.triples, batch)  Triples.negative ←
          worker.SNS.sample_negative(Triples.positive)  /* Compute total loss for the
          batch of triples                                                  */
        Loss_batch ← Total_pairwise_loss(Triples.positive,Triples.positive)  gradients_batch
          ← Optimizer (Loss_batch, worker.model.embeddings)  Gradients ← ∅  /* Use
          All_Reduce to share and collect gradients from all workers for the
          current batch                                                     */
        foreach wrkr ∈ Workers do
            gradient_wrkr ← worker.AllReduce.get_gradients(batch, wrkr)
             Gradients.insert(gradient_wrkr)
        /* Aggregate all collected gradients to update embeddings in the local
          embedding model                                                   */
        gradient ← AllReduce.aggregate_gradients(Gradients)
         worker.model.update_embeddings(gradients)
KGE_trained ← worker.model
```

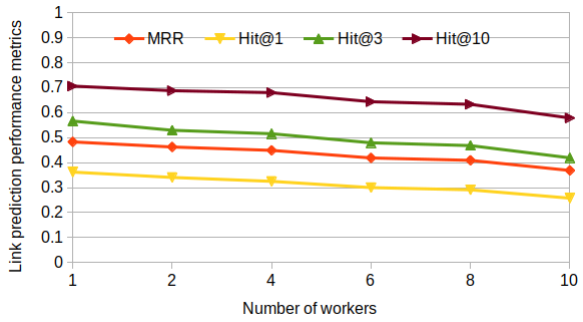
l'optimiseur Adam et nous fixons le taux d'apprentissage à 0,001 et la marge à 4,0.

La Figure 5.3 illustre les métriques de prédiction et le temps de calcul par époque pour l'ensemble de données FB15K. Les méthodes TransE et *DistMult* affichent des valeurs des mesures de performances très proches pour tous les nombres de workers. La solution d'apprentissage centralisée ($worker = 1$) pour FB15K montre les valeurs suivantes des métriques de performances : $MRR \approx 0.5, Hit@10 \approx 0.7, Hit@3 \approx 0.55, Hit@1 \approx 0.35$ (voir figures 5.3(a) et 5.3(b)). Pour l'apprentissage distribué ($workers \geq 2$), les résultats montrent une légère diminution des métriques de performances de prédiction de liens. Cela n'est pas observé avec le plus grand jeu de données YAGO3-10 pour lequel nous voyons que les mesures de performances dans l'apprentissage distribué restent proches de l'apprentissage centralisé. Cela peut être imputé à la différence de taille des jeux de données. Par exemple, YAGO3-10 fait presque le double de taille par rapport à FB15K. En conséquence, la taille de chaque partition est encore suffisamment élevée pour que l'ensemble de données YAGO3-10 apprenne de bons plongements.

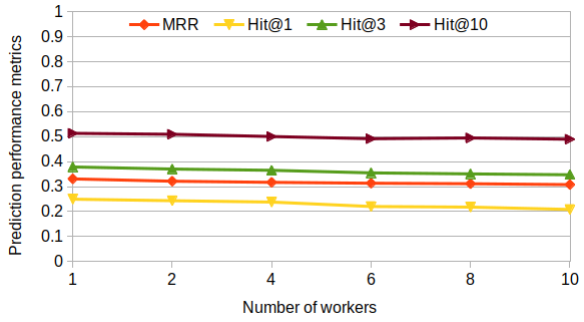
La Figure 5.4 illustre le temps de calcul de notre framework distribué par époque pour



(a)



(b)

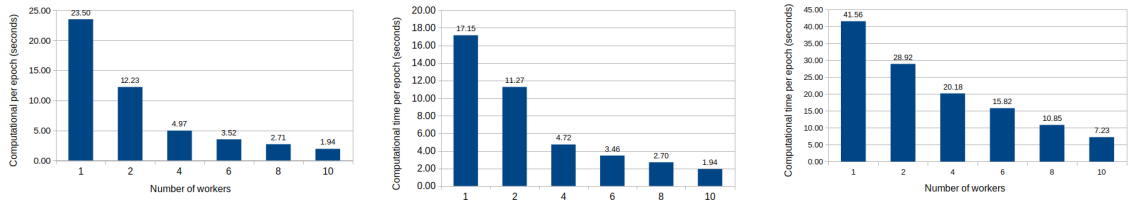


(c)

FIGURE 5.3 – Métriques de performance de prédiction de liens pour les jeux de données FB15K et YAGO pour différents nombres de workers. [Islam, 2022]

différents nombres de workers. On voit clairement que plus le nombre de workers augmente, plus le temps de calcul diminue.

Pour démontrer le grain en temps de calcul, nous présentons les résultats en termes de temps d'exécution dans la Figure 5.5. Nous voyons que le temps d'exécution augmente linéairement pour les modèles de KGE. Nous trouvons le meilleur temps d'exécution pour le modèle TransE sur le jeu de données FB15K. Bien que les métriques de temps d'exécution du modèle *DistMult*



(a) (b) (c)

FIGURE 5.4 – Temps de calcul par époque sur les ensembles de données FB15K et YAGO3-10 pour un nombre différent de workers. [Islam, 2022]

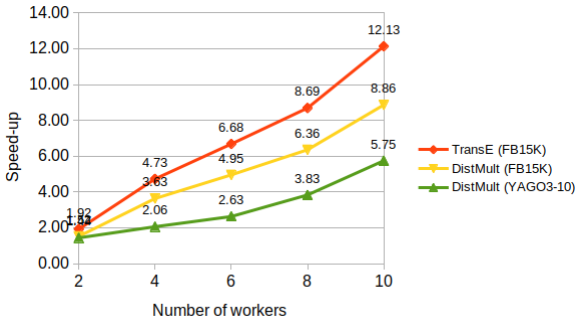


FIGURE 5.5 – Accélération de l'apprentissage distribué de deux modèles de plongement sur les ensembles de données. Les chiffres sur chaque ligne indiquent les mesures d'accélération. [Islam, 2022]

sur le plus grand jeu de données YAGO3-10 soient inférieures à celles du jeu de données FB15K, la métrique de vitesse continue d'augmenter de manière linéaire. Les figures 5.3,5.4 et 5.5 démontrent l'évolutivité de notre framework distribué pour apprendre un modèle géométrique de KGE pour les KG à grande échelle, avec un bon équilibre entre perte de performances et gain de temps.

5.4 Discussion et conclusions

Dans ce chapitre, nous avons présenté notre framework distribué pour le plongement dans des grands graphes de connaissances. Notre framework distribué se distingue des autres frameworks sur quatre points majeurs. Premièrement, dans la littérature et à notre connaissance, la seule tentative d'utilisation de Ray pour le plongement distribuée de KG a été faite par Sheikh *et al.* [Sheikh et al., 2022b] qui prend en charge l'apprentissage distribué de méthodes de plongement basée sur les réseaux de neurones en graphes (GNN). Nous sommes les premiers à utiliser

Ray pour l'apprentissage distribué des approches géométriques de KGE. Deuxièmement, notre framework implémente une méthode SNS qui montre ses performances, alors que tous les autres frameworks utilisent la méthode random-uniform ou l'une de ses variantes pour générer des exemples négatifs. Troisièmement, notre framework utilise l'architecture Ring-AllReduce pour le partage des gradients et la mise à jour des plongements, tandis que le nœud principal collecte les gradients des workers et met à jour le plongement dans la plupart des frameworks existants. Enfin, les frameworks existants utilisent des méthodes de partitionnement de graphes complexes alors que notre framework utilise une méthode de partitionnement aléatoire simple et rapide des graphes de connaissances.

Troisième partie

Projet de recherche scientifique

Chapitre 6

Projet de recherche scientifique

L'ensemble des travaux que j'ai présentés dans ce mémoire d'HDR m'a mené à soulever de nombreuses questions d'avenir dans la continuité de mes travvaux. Ainsi, mes perspectives de recherche dans les prochaines années visent à approfondir l'étude des méthodes d'intelligence artificielle et de calcul distribué, et à les appliquer pour contribuer à la résolution de problèmes biologiques. Plus spécifiquement, je vais me concentrer sur les problématiques suivantes :

1. Méthodes incrémentales d'analyse de graphes dynamiques et leur application à la biologie structurale ;

2. KGE et apprentissage profond à grande échelle pour le repositionnement de médicaments ;

3. Techniques d'apprentissage artificiel avancées pour l'annotation de la fonction des protéines ;

Ces perspectives de recherche sont motivées par ma volonté de contribution à l'avancement des connaissances dans le domaine de la biologie, tout en exploitant le potentiel de l'intelligence artificielle d'aujourd'hui et du calcul distribué pour résoudre des problèmes complexes et ouvrir de nouvelles voies peu explorées.

6.1 Méthodes incrémentales d'analyse de graphes dynamiques et application à la biologie structurale

La biologie structurale est une branche de la biologie moléculaire, de la biochimie et de la biophysique qui s'intéresse à la structure moléculaire des macromolécules biologiques, en particulier les protéines et les acides nucléiques, à la manière dont elles acquièrent les structures qu'elles possèdent et à la manière dont les altérations de leurs structures affectent leur fonction. Ce sujet montre un grand intérêt pour les biologistes, car les macromolécules assurent la plupart des fonctions des cellules, et parce que ce n'est qu'en s'enroulant dans des formes tridimensionnelles spécifiques qu'elles sont capables de remplir ces fonctions [Yang et al., 2022]. La dynamique moléculaire est l'une des techniques les plus utilisées en biologie structurale pour simuler l'évolution (changements conformationnels) de la structure de la protéine au cours du temps. Elle permet de modéliser la trajectoire des particules à partir des forces qui s'exercent entre elles, en utilisant les lois de la physique. En d'autres termes, elle permet de simuler le mouvement des molécules dans un système à l'échelle atomique ou moléculaire, ce qui permet d'étudier les propriétés et le comportement de la matière à un niveau nanométrique. Dans ce contexte, je m'intéresserai, en collaboration avec mes collègues de l'équipe Capsid, à proposer des méthodes à base de graphes pour résoudre certains problèmes en biologie structurale. Un axe de recherche nouveau et important consiste à prédire les sites allostériques dans des protéines importantes [Wu et al., 2022] [Zhu et al., 2022b] [Arantes et al., 2022]. La prédiction des sites allostériques consiste à identifier les régions d'une protéine qui peuvent se lier à des molécules régulatrices, appelées modulateurs allostériques, et qui peuvent ainsi moduler l'activité de la protéine. Ces régions sont souvent situées à distance du site actif de la protéine et peuvent être difficiles à identifier expérimentalement. La dynamique moléculaire est l'une des méthodes utilisées pour prédire ces sites allostériques en simulant le comportement des atomes et des molécules dans la protéine. En utilisant nos approches d'analyse de graphes dynamiques décrites dans les sections 4.3 et 4.4 ainsi que des méthodes de réseaux de neurones en graphes (GNN), je compte analyser les mouvements de la protéine à l'échelle moléculaire et identifier les régions qui sont les plus susceptibles de se lier à des molécules régulatrices. Ces prédictions peuvent ensuite être utilisées pour concevoir de nouveaux médicaments ou pour mieux comprendre les mécanismes de régula-

tion des protéines [Meller et al., 2023].

Un deuxième axe de recherche consiste à modéliser des complexes protéiques, une association de deux ou plusieurs protéines qui interagissent de manière spécifique pour former une structure stable et fonctionnelle, à partir de réseaux PPI dynamiques [Zhang et al., 2016] [Rani et al., 2019]. En effet, ces réseaux PPI sont souvent très complexes et contiennent de nombreuses interactions entre différentes protéines. Elles sont aussi dynamiques et leur nature est souvent hautement dynamique et réactive. Les interactions entre les protéines peuvent être influencées par divers facteurs tels que l'environnement cellulaire, les modifications post-traductionnelles des protéines, les changements conformationnels, etc. Le défi ici est de pouvoir adapter nos méthodes de clustering dynamique à grande échelle [Inoubli et al., 2022] pour prédire les complexes protéiques et de mettre à jour ces complexes en fonction des changements structurels du réseau PPI. D'autres types de méthodes incrémentales d'analyse de graphes peuvent être développées pour analyser et interpréter les réseaux d'interactions protéine-protéine dans la cellule et leur évolution en réponse à une perturbation ou un traitement.

Un dernier axe de recherche consiste à utiliser des méthodes d'analyse de graphes dynamiques pour l'analyse des réseaux d'interactions microbiennes. L'objectif principal est d'utiliser des approches avancées d'apprentissage automatique ainsi que des méthodes à base de graphes afin de prédire la diversité à long terme de l'écosystème microbiologique. En particulier, je viserai à proposer des approches permettant de déduire la diversité directement à partir des propriétés statiques et internes du graphe d'interaction dans lequel les entités sont impliquées. Les approches proposées pourraient être utilisées non seulement pour concevoir des écosystèmes microbiens, mais également pour empêcher le développement d'une souche pathogène dans un écosystème microbien. Les travaux de la thèse de Mohammed Khatbane démarrée en Octobre 2023, en collaboration avec le Laboratoire d'Ingénierie des Biomolécules (LIBio) à Nancy s'inscrivent dans cet axe de recherche.

6.2 KGE et apprentissage profond à grande échelle pour le repositionnement de médicaments

6.2.1 Plongement multivues pour le repositionnement de médicaments

Le plongement multivues a été appliqué avec succès au domaine de la vision par ordinateur où plusieurs vues d'un objet image sont définies pour apprendre un seul plongement de l'objet [Cao et al., 2017, Zhu et al., 2018, Li et al., 2022]. Un plongement multivues consiste en trois étapes principales :

1. identifier plusieurs vues de l'objet ;
2. intégrer l'apprentissage dans chaque vue ;
3. combiner les plongements spécifiques à la vue [Li et al., 2018b].

Le plongement multivues pour le repositionnement de médicaments est une approche utilisée dans le domaine de la découverte de médicaments. Elle vise à prédire les interactions médicament-cible en exploitant des informations à partir de multiples vues ou sources de données. Traditionnellement, la découverte de médicaments implique l'identification de nouvelles molécules qui interagissent de manière spécifique avec une cible biologique donnée, telle qu'une protéine associée à une maladie. Cependant, cette approche peut être coûteuse et prendre beaucoup de temps. Le plongement multivues cherche à résoudre ce problème en utilisant différentes sources d'informations, telles que les données génomiques, les données d'expression génique, les données chimiques et les données de structure protéique. Ces différentes vues fournissent une représentation multidimensionnelle des médicaments et des cibles, ce qui permet d'explorer et d'exploiter les relations complexes entre eux. L'idée principale derrière le plongement multivues est de projeter les médicaments et les cibles dans un espace vectoriel commun, où les similitudes et les relations peuvent être capturées. Cela peut être réalisé en utilisant des techniques d'apprentissage automatique telles que les réseaux de neurones ou les méthodes d'apprentissage par projection. Une fois que les médicaments et les cibles sont représentés dans cet espace commun, il est possible d'utiliser des algorithmes de prédiction de lien pour identifier de nouvelles interactions potentielles et utiles.

En outre, les KG biologiques disponibles contiennent des entités biologiques et différentes

relations entre elles, principalement des interactions entre les biomolécules et leurs fonctions. Cependant, ces KG manquent d'informations structurelles sur les protéines et les composés, tels que les domaines protéiques et les structures chimiques. Une piste de recherche intéressante consiste à inclure des informations structurelles dans ces KG biologiques en plus des informations d'interaction et de fonction pour apprendre de meilleures représentations grâce au plongement multivues. Par conséquent, trois vues pourraient être définies, à savoir la vue structurelle, la vue d'interaction et la vue fonctionnelle. Une méthode de KGE multivues pourrait alors être conçue pour apprendre des plongements de haute qualité dans le contexte d'une tâche de repositionnement de médicaments.

6.2.2 Transférabilité des modèles de KGE

Un paradigme prometteur d'apprentissage automatique efficace est l'apprentissage par transfert, qui se concentre sur le transfert des connaissances acquises dans plusieurs domaines. L'idée générale de l'apprentissage par transfert est d'apprendre un modèle dans un domaine (domaine source) pour une tâche spécifique et de le réutiliser pour des tâches similaires dans des domaines similaires (domaines cibles) afin d'améliorer les performances dans les domaines cibles [Pan and Yang, 2009, Zhuang et al., 2020]. Le paradigme de l'apprentissage par transfert a été étudié et appliqué aux domaines du traitement de l'image et du langage avec des bonnes performances [Liu et al., 2019, Azizpour et al., 2015]. Récemment, peu de chercheurs étudient également la transférabilité ou l'utilisation de modèles pré-entraînés dans des graphes simples [Levie et al., 2021, Qiu et al., 2020, Hu et al., 2019]. L'étude de l'apprentissage par transfert pour les méthodes de KGE pourrait être pertinente dans les KG pour améliorer les performances de prédiction de lien, là où les méthodes de plongement ne fournissent pas de performances satisfaisantes. Dans ce contexte, plusieurs approches d'apprentissage par transfert pour les méthodes de KGE peuvent être utilisées :

— Transfert horizontal (ou intra-tâches) : Cette approche consiste à transférer des connaissances d'une tâche source vers une tâche cible similaire. Par exemple, pour un graphe de connaissances dans un domaine spécifique, il est possible d'entraîner un modèle KGE sur ce graphe, puis transférer les embeddings appris vers un autre graphe de connaissances similaire.

— Transfert vertical (ou inter-tâches) : Le transfert vertical implique de transférer des connaissances d'une tâche source vers une tâche cible qui peut être différente, mais qui partage des similitudes. Par exemple, le transfert de connaissances aprises dans un modèle KGE d'un graphe de connaissances biomédicales vers un graphe de connaissances dans un autre domaine, comme la finance.

— Transfert multi-tâches : Cette approche consiste à entraîner un modèle KGE sur plusieurs tâches sources pour améliorer les performances sur une tâche cible. Par exemple, vous pourriez entraîner un modèle sur plusieurs graphes de connaissances dans des domaines différents, puis utiliser ces embeddings partagés pour améliorer la représentation des entités et des relations dans un nouveau graphe de connaissances.

— Adaptation de domaine (Domain Adaptation) : L'adaptation de domaine vise à adapter les plongements de KG d'un domaine source à un domaine cible qui peut être différent en utilisant des techniques d'adaptation de domaine.

— Ensemble de modèles KGE : il est possible également de combiner les sorties de plusieurs modèles KGE entraînés sur des tâches sources pour améliorer les performances dans la tâche cible. Cette approche peut être particulièrement efficace lorsque les modèles sources sont complémentaires.

6.2.3 Partitionnement intelligent de grands KGs

L'étape de partitionnement d'un KG est une étape importante dans le contexte d'approches distribuées de KGE [Islam et al., 2022]. Il existe quelques méthodes simples de partitionnement de grands graphes dans la littérature, telles que ParMETIS [Karypis and Kumar, 1998c] et KaHIP [Sanders and Schulz, 2011]. Ces méthodes ne peuvent pas être adaptées au partitionnement des graphes de connaissances en raison de trois difficultées majeures. Premièrement, les KG ont des relations nommées entre des paires d'entités. Il est important de prendre en compte les noms de relation lors du partitionnement des KG. Ainsi, ignorer les noms de relation peut entraîner un déséquilibre du nombre de triplets entre différentes partitions pour une certaine relation. Deuxièmement, les KG permettent de multiples relations dirigées entre les paires d'entités. Il est important d'équilibrer les différentes partitions du point de vue du nombre de relations et d'entités et de la connectivité des graphes. Troisièmement, la plupart des KG du monde réel sont

de taille énorme et une seule machine peut ne pas suffire à stocker l'ensemble du KG pendant la tâche de partitionnement. À notre humble connaissance, "Message Cluster and Streaming (MCS)" est la seule méthode pour créer des partitions de KG basées sur le clustering du KG. Cependant, cette méthode néglige toujours la présence de différents noms de relation dans les KG. Par conséquent, le développement d'une méthode de partitionnement de KG est une perspective urgente et potentielle de mes travaux de recherche. Pour cela, je propose d'étudier la possibilité de faire un partitionnement relationnel des triplets ou d'utiliser des ontologies de domaine pour partitionner le KG.

6.2.4 Explicabilité des modèles d'apprentissage profond sur les graphes

L'explicabilité est un aspect important en intelligence artificielle, surtout lorsqu'il s'agit d'algorithmes de réseaux de neurones ou d'apprentissage profond, et en particulier dans le domaine médical. Les architectures d'apprentissage profond produisent des modèles puissants pour de nombreuses tâches et pour plusieurs types de données (séquences, graphes, textes), mais ils sont complexes et difficiles à interpréter [Lisboa et al., 2023]. Par conséquent, il est essentiel de pouvoir expliquer les prédictions et les recommandations de ces modèles pour garantir la transparence et la fiabilité des résultats obtenus. Plusieurs méthodes et approches ont été déjà proposées dans la littérature telle que les méthodes basées sur le mécanisme d'attention, la visualisation, etc. Ces approches sont adoptées aussi pour plusieurs types de données (images, données tabulaires, textes). Dans ce sens, je mettrai l'accent sur l'explicabilité dans les GNNs et les autres algorithmes d'apprentissage profond. De fait, les méthodes d'explicabilté les plus populaires comme LIME [Ribeiro et al., 2016] et SHAP [Lundberg and Lee, 2017] ne traitent que les modèles entraînés sur des images, textes ou données tabulaires et ne traitent pas les GNNs. En outre, L'explicabilité des GNNs attire l'attention d'une large communauté de recherche et quelques algorithmes ont été proposés tels que PGExplainer [Luo et al., 2020], GNNExplainer [Ying et al., 2019] et GraphMask [Schlichtkrull et al., 2022]. Ces algorithmes sont des approches d'explication intrinsèque et donnent des explications généralement de type visualisation. Dans le contexte de cette piste de recherche, j'étudierai comment proposer une méthode d'explicabilité des graphes agnostiques basée sur les règles. En outre, cet axe me permet de renforcer mon équipe de recherche sur le côté explicabilité surtout en lien avec l'application de l'intelligence artificielle dans le domaine de la

santé, qui nécessite généralement des modèles prédictifs explicables vu la sensibilité de données dans ce domaine.

6.3 Techniques d'apprentissage artificiel avancées pour l'annotation de la fonction des protéines

Nos contributions à la recherche dans le domaine de l'annotation automatique des fonctions des protéines consistent principalement à développer des approches computationnelles pour l'annotation fonctionnelle des protéines. Des recherches antérieures ont exploré divers attributs protéiques tels que des séquences, des structures, des domaines, des réseaux d'interaction protéine-protéine, des propriétés physico-chimiques, ainsi que diverses approches informatiques tels que l'extraction de règles d'association, l'apprentissage automatique, l'apprentissage profond, le traitement automatique du langage naturel, l'apprentissage des représentations, l'analyse de réseau, etc. Les contributions présentées dans ce document sont basées sur la structure de graphe. Les résultats expérimentaux montrent une forte association des domaines protéiques et des fonctions, notamment en cas de l'annotation avec des numéros EC (voir Section 1.3).

Dans le chapitre 2.3.3, nous avons présenté nos contributions fondées sur des graphes qui explorent le réseau de protéines pour inférer des annotations fonctionnelles. Au lieu d'un réseau d'interaction protéine-protéine (PPI), le réseau est construit sur une similarité de domaine où les nœuds sont des protéines et les arêtes représentent la similarité dans la composition en domaines des protéines. Une analyse expérimentale approfondie montre des résultats prometteurs. Les résultats de *GrAPFI* et de *GrAPFI-GO* sont facilement explicables car elles emploient une fonction de score interprétable. Néanmoins, les approches proposées ont certaines limites. Il existe de nombreuses séquences protéiques courtes en termes de nombre d'acides aminés qui les composent pour lesquelles les domaines ne sont pas disponibles. Dans de tels cas, nos contributions sont limitées pour trouver une annotation, car les protéines ne peuvent pas être liées au graphe de protéines sous-jacent. L'un des inconvénients des approches *GrAPFI* et *GrAPFI-GO* est qu'elles fonctionnent seulement sur des domaines protéiques. Cependant, il existe de nombreux autres attributs tels que la voie métabolique, le génotype, le phénotype et le taxon qui pourraient incorporer des informations importantes dans le processus. De plus, les relations hiérarchiques entre

les attributs représentés sous forme d'ontologie peuvent constituer un ajout significatif au processus. Dans la Section 2.4, nous avons discuté l'utilisation des graphes de connaissances pour intégrer diverses sources de données et de les utiliser pour l'annotation des protéines. Dans ce contexte, nous avons commencé à examiner et explorer de nouvelles pistes de recherche combinant des techniques d'apprentissage artificiel avancées avec des graphes de protéines ou des graphes de connaissances biologiques dédiés à la tâche d'annotation fonctionnelle de protéines. Dans un premier travail exploratoire, nous avons généré un graphe de connaissances biologique dédié à cette tâche d'annotation fonctionnelle de protéines et nous avons défini une méthode basée sur les réseaux antagonistes génératifs (GAN) pour la prédiction des annotations de type GO. En suivant les terminologies du GAN, nous formons d'abord un discriminateur en utilisant un échantillonnage négatif adaptatif au domaine pour discriminer les triplets positifs et négatifs, puis nous formons un générateur pour guider une marche aléatoire dans le graphe de connaissances qui identifie les chemins entre les protéines et les annotations GO [Sarker et al., 2021].

Le deuxième travail exploratoire consiste à proposer un nouveau modèle d'apprentissage profond semi-supervisé qui vise à apprendre de meilleures représentations latentes pour chaque protéine/nœud en prenant en compte les informations de voisinage afin d'améliorer l'annotation. Tout d'abord, nous extrayons un ensemble de caractéristiques à partir de données de protéines brutes. Chaque protéine est associée à un vecteur caractéristique 1-D qui représente sa composition en domaines InterPro. Comme le nombre de domaines InterPro possibles est très élevé, nous avons conçu un modèle d'auto-encodeur profond (DAE) qui cherche à trouver une représentation efficace de la composition en domaines des protéines dans un espace latent de dimension inférieure. Ensuite, nous avons construit un graphe de protéines où chaque nœud est une protéine associée à son vecteur de représentation latente et chaque arête est pondérée par la distance euclidienne entre les deux nœuds qu'elle relie. Enfin, nous avons formé un réseau neuronal de graphe semi-supervisé (SGNN) pour l'annotation automatique de la fonction des protéines à l'aide du graphe de protéines construit [Sellami et al., 2022]. Le modèle a été testé sur quatre protéomes de référence dans UniProtKB/SwissProt, à savoir Human, Arabidopsis Thaliana, Souris et Rat. Les résultats expérimentaux montrent que le modèle proposé est compétitif pour l'annotation fonctionnelle des protéines par rapport aux méthodes existantes

Nous allons continuer nos réflexions sur l'utilisation de techniques avancées d'apprentissage

artificiel en essayant d'enrichir le modèle SGNN par l'intégration de nouvelles caractéristiques afin d'optimiser l'annotation fonctionnelle de protéines. Une piste prometteuse pour améliorer l'annotation fonctionnelle des protéines consiste à intégrer des informations sur leur structure tridimensionnelle dans les graphes multivues [Yan et al., 2021b]. Les graphes multivues peuvent être utilisés pour intégrer des informations sur la structure tridimensionnelle des protéines dans le modèle SGNN afin d'optimiser l'annotation fonctionnelle des protéines. Les informations structurales telles que la forme de la protéine et la position des résidus peuvent fournir des indices cruciaux sur leur fonction et leur interaction avec d'autres protéines. Par exemple, les interactions entre les domaines de protéines différentes peuvent être liées à leur orientation et à leur proximité spatiale. En incorporant ces informations dans les modèles d'apprentissage multivues sur les graphes, nous pourrions améliorer considérablement la précision de l'annotation fonctionnelle. En effet, cela permettrait d'explorer des caractéristiques de la structure tridimensionnelle qui ne sont pas accessibles avec les données de séquence seule, et ainsi de mieux comprendre la relation entre la structure et la fonction des protéines. Cependant, l'intégration de données structurales dans les graphes multivues est un défi important car elle nécessite la mise en place d'une représentation adéquate des données tridimensionnelles dans un espace multivues. De plus, cela nécessite une manipulation complexe de données volumineuses, ce qui peut rendre les calculs extrêmement coûteux en termes de temps et de ressources. Malgré ces défis, l'incorporation de données structurales dans les graphes multivues est une piste prometteuse pour l'amélioration de l'annotation fonctionnelle des protéines. Cette approche pourrait aider à améliorer notre compréhension des mécanismes moléculaires sous-jacents à la fonction des protéines, ainsi que de faciliter la découverte de nouveaux médicaments ciblant des protéines spécifiques.

Une autre perspective intéressante pour améliorer la précision de l'annotation fonctionnelle des protéines est d'incorporer des informations sur leur expression dans différents tissus. En effet, l'expression des protéines varie en fonction du tissu et de l'état physiologique, ce qui peut affecter leur fonction. Pour cela, on pourrait utiliser des données d'expression génique pour annoter les protéines en fonction de leur expression tissulaire. Pour intégrer ces informations dans notre modèle de graphe multivues, nous pourrions créer des vues supplémentaires pour chaque tissu, où chaque nœud représente une protéine et les arêtes sont pondérées en fonction de la corrélation d'expression entre les protéines dans chaque tissu. Ensuite, nous pourrions combiner

ces vues avec notre vue existante qui utilise des informations sur les domaines de protéines pour créer un graphe multivues complet. Nous pourrions ensuite former un modèle d'apprentissage en utilisant ce graphe multivues et les annotations fonctionnelles connues pour prédire les fonctions des protéines inconnues. En utilisant cette approche, nous pourrions améliorer la précision de l'annotation fonctionnelle des protéines en intégrant des informations sur leur expression dans différents tissus.

En complément des perspectives mentionnées précédemment, il est intéressant d'explorer d'autres modèles d'apprentissage automatique pour améliorer la précision de l'annotation fonctionnelle des protéines. Par exemple, les réseaux de neurones convolutifs (CNNs) sont couramment utilisés pour l'analyse de séquences biologiques. Ils sont particulièrement utiles pour extraire des caractéristiques locales dans les séquences de protéines, ce qui peut être utile pour l'annotation fonctionnelle. En incorporant des informations de séquence ainsi que des données d'expression de tissus dans un modèle de réseaux de neurones convolutif, il est possible d'améliorer la précision de l'annotation fonctionnelle. D'autre part, les réseaux de neurones récurrents (RNNs) peuvent être utilisés pour analyser des séquences biologiques de longueur variable [Kavipriya and Manjula, 2023]. Les RNNs sont capables de capturer les dépendances à long terme dans les séquences, ce qui peut être particulièrement utile pour l'annotation fonctionnelle des protéines. En utilisant des modèles de réseaux de neurones récurrents, il est possible de prendre en compte l'historique des domaines InterPro dans la séquence protéique, ce qui peut améliorer la précision de l'annotation fonctionnelle.

Bibliographie

[Anil et al., 2020] Anil, R., Capan, G., Drost-Fromm, I., Dunning, T., Friedman, E., Grant, T., Quinn, S., Ranjan, P., Schelter, S., and Yilmazel, O. (2020). Apache mahout : Machine learning on distributed dataflow systems. *J. Mach. Learn. Res.*, 21(1).

[Arantes et al., 2022] Arantes, P. R., Patel, A. C., and Palermo, G. (2022). Emerging methods and applications to decrypt allostery in proteins and nucleic acids. *Journal of Molecular Biology*, 434(17) :167518. Allostery : From Mechanisms to Therapies.

[Aridhi et al., 2016a] Aridhi, S., Brugnara, M., Montresor, A., and Velegrakis, Y. (2016a). Distributed k-core decomposition and maintenance in large dynamic graphs. In Gal, A., Weidlich, M., Kalogeraki, V., and Venkasubramanian, N., editors, *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS '16, Irvine, CA, USA, June 20 - 24, 2016*, pages 161–168. ACM.

[Aridhi and Mephu, 2016] Aridhi, S. and Mephu, E. (2016). Big Graph Mining : Frameworks and Techniques. *Big Data Research*, 6 :1–10.

[Aridhi et al., 2017] Aridhi, S., Montresor, A., and Velegrakis, Y. (2017). Bladyg : A graph processing framework for large dynamic graphs. *Big Data Research*, 9 :9–17.

[Aridhi et al., 2016b] Aridhi, S., Sghaier, H., Zoghlami, M., Maddouri, M., and Nguifo, E. M. (2016b). Prediction of ionizing radiation resistance in bacteria using a multiple instance learning model. *J. Comput. Biol.*, 23(1) :10–20.

[Ashburner and et al., 2000] Ashburner, M. and et al. (2000). Gene ontology : tool for the unification of biology. *Nature genetics*, 25(1) :25.

[Azizpour et al., 2015] Azizpour, H., Razavian, A. S., Sullivan, J., Maki, A., and Carlsson, S.

(2015). Factors of transferability for a generic convnet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(9) :1790–1802.

[Bajaber et al., 2016] Bajaber, F., Elshawi, R., Batarfi, O., Altalhi, A., Barnawi, A., and Sakr, S. (2016). Big data 2.0 processing systems : Taxonomy and open challenges. *Journal of Grid Computing*, 14(3) :379–405.

[Balouek et al., 2013] Balouek, D., Carpen Amarie, A., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lèbre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Pérez, C., Quesnel, F., Rohr, C., and Sarzyniec, L. (2013). Adding virtualization capabilities to the Grid'5000 testbed. In Ivanov, I. I., van Sinderen, M., Leymann, F., and Shan, T., editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer.

[Berman et al., 2000] Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., and Bourne, P. E. (2000). The Protein Data Bank. *Nucleic Acids Research*, 28(1) :235–242.

[Blokdyk, 2020] Blokdyk, G. (2020). *Apache NiFi A Complete Guide - 2020 Edition*. Emereo Pty Limited.

[Bordes et al., 2013] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2*, pages 2787–2795.

[Bruck and Ho, 1993] Bruck, J. and Ho, C.-T. (1993). Efficient global combine operations in multi-port message-passing systems. *Parallel Processing Letters*, 3(04) :335–346.

[Cai et al., 2003] Cai, C., Han, L., Ji, Z. L., Chen, X., and Chen, Y. Z. (2003). Svm-prot : web-based support vector machine software for functional classification of a protein from its primary sequence. *Nucleic acids research*, 31(13) :3692–3697.

[Cai and Wang, 2018] Cai, L. and Wang, W. Y. (2018). KBGAN : Adversarial learning for knowledge graph embeddings. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long Papers)*, pages 1470–1480.

[Cao et al., 2017] Cao, G., Iosifidis, A., Chen, K., and Gabbouj, M. (2017). Generalized multi-view embedding for visual recognition and cross-modal retrieval. *IEEE Transactions on Cybernetics*, 48(9) :2542–2555.

[Chambers et al., 2010] Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R. R., Bradshaw, R., and Weizenbaum, N. (2010). Flumejava : Easy, efficient data-parallel pipelines. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '10, page 363–375, New York, NY, USA. Association for Computing Machinery.

[Choudhary et al., 2021] Choudhary, N., Rao, N., Katariya, S., Subbian, K., and Reddy, C. (2021). Probabilistic entity representation model for reasoning over knowledge graphs. In *Advances in Neural Information Processing Systems*, volume 34, pages 23440–23451.

[Consortium, 2015] Consortium, T. U. (2015). Uniprot : a hub for protein information. *Nucleic Acids Research*, 43(D204-D212).

[Cornish-Bowden, 2014] Cornish-Bowden, A. (2014). Current iubmb recommendations on enzyme nomenclature and kinetics. *Perspectives in Science*, 1(1-6) :74–87.

[Coulet, 2019] Coulet, A. (2019). *Mises en correspondances de données, textes et connaissances pour la découverte de connaissances biomédicales.* Habilitation à diriger des recherches, Université de Lorraine.

[Dalkiran et al., 2018] Dalkiran, A., Rifaioglu, A. S., Martin, M. J., Cetin-Atalay, R., Atalay, V., and Doğan, T. (2018). ECPred : a tool for the prediction of the enzymatic functions of protein sequences based on the EC nomenclature. *BMC Bioinformatics*, 19(1) :334.

[De Francisci Morales, 2013] De Francisci Morales, G. (2013). Samoa : A platform for mining big data streams. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, page 777–778, New York, NY, USA. Association for Computing Machinery.

[Fayyad et al., 1996] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11) :27–34.

[Fiscon et al., 2021] Fiscon, G., Conte, F., Farina, L., and Paci, P. (2021). Saverunner : A network-based algorithm for drug repurposing and its application to COVID-19. *PLoS Computational Biology*, 17(2) :e1008686.

[Gao et al., 2022] Gao, Z., Ding, P., and Xu, R. (2022). Kg-predict : A knowledge graph computational framework for drug repurposing. *Journal of Biomedical Informatics*, 132 :104133.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

[Google, 2013] Google (2013). Freebase data dumps. `https://developers.google.com/freebase/data`.

[Gurung, 2020] Gurung, A. B. (2020). In silico structure modelling of sars-cov-2 nsp13 helicase and nsp14 and repurposing of fda approved antiviral drugs as dual inhibitors. *Gene Reports*, 21 :100860.

[Hu et al., 2019] Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. (2019). Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*.

[Inoubli, 2021] Inoubli, W. (2021). *Analysis and Mining of Large Dynamic Graphs : case of graph clustering. (Fouille et Analyse de Grands Graphes Dynamiques : Application dans le clustering de graphes)*. PhD thesis, Tunis El Manar University, Tunisia.

[Inoubli et al., 2018] Inoubli, W., Aridhi, S., Mezni, H., Maddouri, M., and Nguifo, E. M. (2018). An experimental survey on big data frameworks. *Future Gener. Comput. Syst.*, 86 :546–564.

[Inoubli et al., 2022] Inoubli, W., Aridhi, S., Mezni, H., Maddouri, M., and Nguifo, E. M. (2022). A distributed and incremental algorithm for large-scale graph clustering. *Future Gener. Comput. Syst.*, 134 :334–347.

[Inoubli et al., 2017] Inoubli, W., Cruz, L. A., da Silva, T. L. C., Coutinho, G., Peres, L., Magalhães, R. P., de Macêdo, J. A. F., Aridhi, S., and Nguifo, E. M. (2017). A distributed framework for large-scale time-dependent graph analysis. In Aridhi, S., de Macêdo, J. A. F., Nguifo, E. M., and Zeitouni, K., editors, *Proceedings of the Workshop on Large-Scale Time Dependent Graphs (TD-LSG 2017) co-located with the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2017),*

Skopje, Macedonia, September 18, 2017, volume 1929 of *CEUR Workshop Proceedings*, pages 48–53. CEUR-WS.org.

[Ioannidis et al., 2020] Ioannidis, V. N., Song, X., Manchanda, S., Li, M., Pan, X., Zheng, D., Ning, X., Zeng, X., and Karypis, G. (2020). Drkg-drug repurposing knowledge graph for COVID-19. *Github* `https://github.com/gnn4dr/DRKG`.

[Islam, 2022] Islam, K. (2022). *Explainable link prediction in large complex graphs - application to drug repurposing*. Theses, Université de Lorraine.

[Islam et al., 2023] Islam, M. K., Amaya-Ramirez, D., Maigret, B., Devignes, M.-D., Aridhi, S., and Smaïl-Tabbone, M. (2023). Molecular-evaluated and explainable drug repurposing for COVID-19 using ensemble knowledge graph embedding. *Scientific Reports*, 13(1) :3643.

[Islam et al., 2021] Islam, M. K., Aridhi, S., and Smaïl-Tabbone, M. (2021). Simple negative sampling for link prediction in knowledge graphs. In Benito, R. M., Cherifi, C., Cherifi, H., Moro, E., Rocha, L. M., and Sales-Pardo, M., editors, *Complex Networks & Their Applications X - Volume 2, Proceedings of the Tenth International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2021, Madrid, Spain, November 30 - December 2, 2021*, volume 1016 of *Studies in Computational Intelligence*, pages 549–562. Springer.

[Islam et al., 2022] Islam, M. K., Aridhi, S., and Smaïl-Tabbone, M. (2022). Negative sampling and rule mining for explainable link prediction in knowledge graphs. *Knowl. Based Syst.*, 250 :109083.

[Ji et al., 2015] Ji, G., He, S., Xu, L., Liu, K., and Zhao, J. (2015). Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1 : Long papers)*, pages 687–696.

[Jiang and et al., 2016] Jiang, Y. and et al. (2016). An expanded evaluation of protein function prediction methods shows an improvement in accuracy. *Genome biology*, 17(1) :184.

[Jones et al., 2014] Jones, P., Binns, D., Chang, H.-Y., Fraser, M., Li, W., McAnulla, C., McWilliam, H., Maslen, J., Mitchell, A., Nuka, G., Pesseat, S., Quinn, A. F., Sangrador-Vegas, A., Scheremetjew, M., Yong, S.-Y., Lopez, R., and Hunter, S. (2014). InterProScan 5 : genome-scale protein function classification. *Bioinformatics*, 30(9) :1236–1240.

[Kanatsoulis and Sidiropoulos, 2021] Kanatsoulis, C. I. and Sidiropoulos, N. D. (2021). Texgraph : Coupled tensor-matrix knowledge-graph embedding for COVID-19 drug repurposing. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 603–611. SIAM.

[Karypis and Kumar, 1998a] Karypis, G. and Kumar, V. (1998a). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1) :359–392.

[Karypis and Kumar, 1998b] Karypis, G. and Kumar, V. (1998b). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1) :359–392.

[Karypis and Kumar, 1998c] Karypis, G. and Kumar, V. (1998c). A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of parallel and distributed computing*, 48(1) :71–95.

[Kavipriya and Manjula, 2023] Kavipriya, G. and Manjula, D. (2023). Drug–target interaction prediction model using optimal recurrent neural network. *Intelligent Automation & Soft Computing*, 35(2).

[Klyne, 2004] Klyne, G. (2004). Resource description framework (RDF) : Concepts and abstract syntax. *W3C Recommendation*.

[Kok and Domingos, 2007] Kok, S. and Domingos, P. (2007). Statistical predicate invention. In *Proceedings of the 24th International Conference on Machine Learning*, pages 433–440.

[Kovacs et al., 2019] Kovacs, I. A., Luck, K., Spirohn, K., Wang, Y., Pollis, C., Schlabach, S., Bian, W., Kim, D.-K., Kishore, N., Hao, T., et al. (2019). Network-based prediction of protein interactions. *Nature Communications*, 10(1) :1–8.

[Kulmanov and Hoehndorf, 2020] Kulmanov, M. and Hoehndorf, R. (2020). Deepgoplus : improved protein function prediction from sequence. *Bioinformatics*, 36(2) :422–429.

[Kummerfeld and Teichmann, 2009] Kummerfeld, S. K. and Teichmann, S. A. (2009). Protein domain organisation : adding order. *BMC Bioinform.*, 10.

[Leggas et al., 2021] Leggas, D., Baskaran, M., Ezick, J., and von Hofe, B. (2021). Filtered

tensor construction and decomposition for drug repositioning. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE.

[Lerer et al., 2019] Lerer, A., Wu, L., Shen, J., Lacroix, T., Wehrstedt, L., Bose, A., and Peysakhovich, A. (2019). Pytorch-biggraph : A large scale graph embedding system. *Proceedings of Machine Learning and Systems*, 1 :120–131.

[Levie et al., 2021] Levie, R., Huang, W., Bucci, L., Bronstein, M., and Kutyniok, G. (2021). Transferability of spectral graph convolutional neural networks. *Journal of Machine Learning Research*, 22(272) :1–59.

[Li et al., 2014] Li, M., Andersen, D. G., Smola, A. J., and Yu, K. (2014). Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems*, 27.

[Li et al., 2018a] Li, Y., Wang, S., Umarov, R., Xie, B., Fan, M., Li, L., and Gao, X. (2018a). DEEPre : sequence-based enzyme EC number prediction by deep learning. *Bioinformatics*, 34(5) :760–769.

[Li et al., 2018b] Li, Y., Yang, M., and Zhang, Z. (2018b). A survey of multi-view representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 31(10) :1863–1883.

[Li et al., 2022] Li, Z., Guo, C., Feng, Z., Hwang, J.-N., and Xue, X. (2022). Multi-view visual semantic embedding. In *International Joint Conferences on Artificial Intelligence*.

[Liang et al., 2022] Liang, Z., Yang, J., Liu, H., Huang, K., Cui, L., Qu, L., and Li, X. (2022). Hrer : A new bottom-up rule learning for knowledge graph completion. *Electronics*, 11(6) :908.

[Lisboa et al., 2023] Lisboa, P., Saralajew, S., Vellido, A., Fernández-Domenech, R., and Villmann, T. (2023). The coming of age of interpretable and explainable machine learning models. *Neurocomputing*, 535 :25–39.

[Liu et al., 2019] Liu, N. F., Gardner, M., Belinkov, Y., Peters, M. E., and Smith, N. A. (2019). Linguistic knowledge and transferability of contextual representations. In *Proceedings of the 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1073–1094.

[Low et al., 2012] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., and Hellerstein,

J. M. (2012). Distributed graphlab : A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8) :716–727.

[Lü and Zhou, 2011] Lü, L. and Zhou, T. (2011). Link prediction in complex networks : A survey. *Physica A : Statistical Mechanics and Its Applications*, 390(6) :1150–1170.

[Lundberg and Lee, 2017] Lundberg, S. and Lee, S.-I. (2017). A unified approach to interpreting model predictions.

[Luo et al., 2020] Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., and Zhang, X. (2020). Parameterized explainer for graph neural network.

[Malewicz et al., 2010] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel : a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM.

[Meller et al., 2023] Meller, A., Ward, M., Borowsky, J., Kshirsagar, M., Lotthammer, J. M., Oviedo, F., Ferres, J. L., and Bowman, G. R. (2023). Predicting locations of cryptic pockets from single protein structures using the pocketminer graph neural network. *Nature Communications*, 14(1) :1177.

[Menon and Elkan, 2011] Menon, A. K. and Elkan, C. (2011). Link prediction via matrix factorization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 437–452. Springer.

[Min et al., 2013] Min, B., Grishman, R., Wan, L., Wang, C., and Gondek, D. (2013). Distant supervision for relation extraction with an incomplete knowledge base. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, pages 777–782.

[Mohamed et al., 2020] Mohamed, S. K., Novavcek, V., and Nounu, A. (2020). Discovering protein drug targets using knowledge graph embeddings. *Bioinformatics*, 36(2) :603–610.

[Moritz et al., 2018a] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., et al. (2018a). Ray : A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577.

[Moritz et al., 2018b] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., and Stoica, I. (2018b). Ray : A distributed framework for emerging ai applications. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 561–577, USA. USENIX Association.

[Narkhede et al., 2017] Narkhede, N., Shapira, G., and Palino, T. (2017). *Kafka : The Definitive Guide Real-Time Data and Stream Processing at Scale*. O'Reilly Media, Inc., 1st edition.

[Pan and Yang, 2009] Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10) :1345–1359.

[Paulheim, 2017] Paulheim, H. (2017). Knowledge graph refinement : A survey of approaches and evaluation methods. *Semantic Web*, 8(3) :489–508.

[Perez-Lemus et al., 2022] Perez-Lemus, G. R., Menendez, C. A., Alvarado, W., Bylehn, F., and de Pablo, J. J. (2022). Toward wide-spectrum antivirals against coronaviruses : Molecular characterization of sars-cov-2 nsp13 helicase inhibitors. *Science advances*, 8(1) :eabj4526.

[Qiu et al., 2020] Qiu, J., Chen, Q., Dong, Y., Zhang, J., Yang, H., Ding, M., Wang, K., and Tang, J. (2020). Gcc : Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1150–1160.

[Radivojac and et al., 2013] Radivojac, P. and et al. (2013). A large-scale evaluation of computational protein function prediction. *Nature methods*, 10(3) :221.

[Rani et al., 2019] Rani, R. R., Ramyachitra, D., and Brindhadevi, A. (2019). Detection of dynamic protein complexes through markov clustering based on elephant herd optimization approach. *Scientific Reports*, 9(1) :11106.

[Ribeiro et al., 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should i trust you ?" : Explaining the predictions of any classifier.

[Rossi et al., 2021] Rossi, A., Barbosa, D., Firmani, D., Matinata, A., and Merialdo, P. (2021). Knowledge graph embedding for link prediction : A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2) :1–49.

[Roy et al., 2012] Roy, A., Yang, J., and Zhang, Y. (2012). Cofactor : an accurate comparative algorithm for structure-based protein function annotation. *Nucleic acids research*, 40(W1) :W471–W477.

[Sadeghian et al., 2019] Sadeghian, A., Armandpour, M., Ding, P., and Wang, D. Z. (2019). Drum : end-to-end differentiable rule mining on knowledge graphs. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 15347–15357.

[Sakouhi et al., 2016] Sakouhi, C., Aridhi, S., Guerrieri, A., Sassi, S., and Montresor, A. (2016). Dynamicdfep : A distributed edge partitioning approach for large dynamic graphs. In Desai, E., Desai, B. C., Toyama, M., and Bernardino, J., editors, *Proceedings of the 20th International Database Engineering & Applications Symposium, IDEAS 2016, Montreal, QC, Canada, July 11-13, 2016*, pages 142–147. ACM.

[Sanders and Schulz, 2011] Sanders, P. and Schulz, C. (2011). Engineering multilevel graph partitioning algorithms. In *European Symposium on Algorithms*, pages 469–480. Springer.

[Sarker, 2021] Sarker, B. (2021). *On Graph-Based Approaches for Protein Function Annotation and Knowledge Discovery*. Theses, Université de Lorraine.

[Sarker et al., 2021] Sarker, B., Devignes, M.-D., Wolf, G., and Aridhi, S. (2021). Prot-A-GAN : Automatic Protein Function Annotation using GAN-inspired Knowledge Graph Embedding. In *ICML 2021 - Workshop on Computational Biology*, Virtual, United States.

[Sarker et al., 2020a] Sarker, B., Khare, N., Devignes, M., and Aridhi, S. (2020a). Graph based automatic protein function annotation improved by semantic similarity. In Rojas, I., Valenzuela, O., Rojas, F., Herrera, L. J., and Guzman, F. M. O., editors, *Bioinformatics and Biomedical Engineering - 8th International Work-Conference, IWBBIO 2020, Granada, Spain, May 6-8, 2020, Proceedings*, volume 12108 of *Lecture Notes in Computer Science*, pages 261–272. Springer.

[Sarker et al., 2022] Sarker, B., Khare, N., Devignes, M., and Aridhi, S. (2022). Improving automatic GO annotation with semantic similarity. *BMC Bioinform.*, 23-S(2) :433.

[Sarker et al., 2020b] Sarker, B., Ritchie, D. W., and Aridhi, S. (2020b). Grapfi : predicting enzymatic function of proteins from domain similarity graphs. *BMC Bioinform.*, 21(1) :168.

[Schlichtkrull et al., 2022] Schlichtkrull, M. S., Cao, N. D., and Titov, I. (2022). Interpreting graph neural networks for nlp with differentiable edge masking.

[Sellami et al., 2022] Sellami, A., Sarker, B., Tabbone, S., Devignes, M.-D., and Aridhi, S. (2022). A semi-supervised graph deep neural network for automatic protein function annotation. In Rojas, I., Valenzuela, O., Rojas, F., Herrera, L. J., and Ortuño, F., editors, *Bioinformatics and Biomedical Engineering*, pages 153–166, Cham. Springer International Publishing.

[Sergeev and Del Balso, 2018] Sergeev, A. and Del Balso, M. (2018). Horovod : fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv :1802.05799*.

[Shao et al., 2013] Shao, B., Wang, H., and Li, Y. (2013). Trinity : A distributed graph engine on a memory cloud. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, page 505–516, New York, NY, USA. Association for Computing Machinery.

[Sheikh et al., 2022a] Sheikh, N., Qin, X., Gur, Y., Reinwald, B., Xiang, Q., Yu, H., et al. (2022a). Distributed training of knowledge graph embedding models using ray. In *Twenty-fifth International Conference on Extending Database Technology*, pages 2–549.

[Sheikh et al., 2022b] Sheikh, N., Qin, X., Reinwald, B., and Lei, C. (2022b). Scaling knowledge graph embedding models. *arXiv preprint arXiv :2201.02791*.

[Singh, 2016] Singh, S. (2016). Cluster-level logging of containers with containers : Logging challenges of container-based cloud deployments. *Queue*, 14(3) :83–106.

[Somolinos et al., 2021] Somolinos, F. J., León, C., and Guerrero-Aspizua, S. (2021). Drug repurposing using biological networks. *Processes*, 9(6) :1057.

[Sosa et al., 2019] Sosa, D. N., Derry, A., Guo, M., Wei, E., Brinton, C., and Altman, R. B. (2019). A literature-based knowledge graph embedding method for identifying drug repurposing opportunities in rare diseases. In *Pacific Symposium on Biocomputing 2020*, pages 463–474. World Scientific.

[Sun et al., 2018] Sun, Z., Deng, Z.-H., Nie, J.-Y., and Tang, J. (2018). Rotate : Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*.

[Thakur et al., 2005] Thakur, R., Rabenseifner, R., and Gropp, W. (2005). Optimization of collective communication operations in mpich. *The International Journal of High Performance Computing Applications*, 19(1) :49–66.

[Tian et al., 2013] Tian, Y., Balmin, A., Corsten, S. A., Tatikonda, S., and McPherson, J. (2013). From "think like a vertex" to "think like a graph". *Proc. VLDB Endow.*, 7(3) :193–204.

[Trouillon et al., 2016] Trouillon, T., Welbl, J., Riedel, S., Gaussier, E., and Bouchard, G. (2016). Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080. PMLR.

[Veras et al., 2022] Veras, M. B. A., Sarker, B., Aridhi, S., Gomes, J. P. P., de Macêdo, J. A. F., Nguifo, E. M., Devignes, M., and Smaïl-Tabbone, M. (2022). On the design of a similarity function for sparse binary data with application on protein function annotation. *Knowl. Based Syst.*, 238 :107863.

[Vivek-Ananth et al., 2022] Vivek-Ananth, R., Krishnaswamy, S., and Samal, A. (2022). Potential phytochemical inhibitors of sars-cov-2 helicase nsp13 : a molecular docking and dynamic simulation study. *Molecular Diversity*, 26(1) :429–442.

[Wang et al., 2021] Wang, M., Qiu, L., and Wang, X. (2021). A survey on knowledge graph embeddings for link prediction. *Symmetry*, 13(3) :485.

[Wang et al., 2017] Wang, Q., Mao, Z., Wang, B., and Guo, L. (2017). Knowledge graph embedding : A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12) :2724–2743.

[Wang et al., 2014] Wang, Z., Zhang, J., Feng, J., and Chen, Z. (2014). Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

[White et al., 2020] White, M. A., Lin, W., and Cheng, X. (2020). Discovery of COVID-19 inhibitors targeting the sars-cov-2 nsp13 helicase. *The Journal of Physical Chemistry Letters*, 11(21) :9144–9151.

[Wu et al., 2022] Wu, N., Strömich, L., and Yaliraki, S. N. (2022). Prediction of allosteric sites and signaling : Insights from benchmarking datasets. *Patterns*, 3(1) :100408.

[Xin et al., 2013] Xin, R. S., Gonzalez, J. E., Franklin, M. J., and Stoica, I. (2013). Graphx : a resilient distributed graph system on spark. In Boncz, P. A. and Neumann, T., editors, *GRADES*, page 2. CWI/ACM.

[Xu et al., 2007] Xu, X., Yuruk, N., Feng, Z., and Schweiger, T. A. J. (2007). Scan : a structural clustering algorithm for networks. In *KDD '07 : Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833, New York, NY, USA. ACM.

[Xu et al., 2016] Xu, Z., Pu, C., and Yang, J. (2016). Link prediction based on path entropy. *Physica A : Statistical Mechanics and its Applications*, 456 :294–301.

[Yan et al., 2014] Yan, D., Cheng, J., Lu, Y., and Ng, W. (2014). Blogel : A block-centric framework for distributed computation on real-world graphs. *Proc. VLDB Endow.*, 7(14) :1981–1992.

[Yan et al., 2021a] Yan, V. K., Li, X., Ye, X., Ou, M., Luo, R., Zhang, Q., Tang, B., Cowling, B. J., Hung, I., Siu, C. W., et al. (2021a). Drug repurposing for the treatment of COVID-19 : A knowledge graph approach. *Advanced Therapeutics*, 4(7) :2100055.

[Yan et al., 2021b] Yan, X., Hu, S., Mao, Y., Ye, Y., and Yu, H. (2021b). Deep multi-view learning methods : A review. *Neurocomputing*, 448 :106–129.

[Yang et al., 2015] Yang, B., Yih, S. W.-t., He, X., Gao, J., and Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations*.

[Yang and Amazon, 2018] Yang, C. and Amazon, A. (2018). Tree-based allreduce communication on mxnet. *Technical Report*.

[Yang et al., 2022] Yang, C., Chen, E. A., and Zhang, Y. (2022). Proteinndash ;ligand docking in the machine-learning era. *Molecules*, 27(14).

[Ying et al., 2019] Ying, R., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. (2019). *GNNExplainer : Generating Explanations for Graph Neural Networks*. Curran Associates Inc., Red Hook, NY, USA.

[Zhang et al., 2018] Zhang, C., Zheng, W., Freddolino, P. L., and Zhang, Y. (2018). Metago : Predicting gene ontology of non-homologous proteins through low-resolution protein structure

prediction and protein–protein network mapping. *Journal of molecular biology*, 430(15) :2256–2265.

[Zhang et al., 2021] Zhang, R., Hristovski, D., Schutte, D., Kastrin, A., Fiszman, M., and Kilicoglu, H. (2021). Drug repurposing for COVID-19 via knowledge graph completion. *Journal of Biomedical Informatics*, 115 :103696.

[Zhang et al., 2016] Zhang, Y., Lin, H., Yang, Z., Wang, J., Liu, Y., and Sang, S. (2016). A method for predicting protein complex in dynamic ppi networks. *BMC Bioinformatics*, 17(7) :229.

[Zhang et al., 2019] Zhang, Y., Yao, Q., Shao, Y., and Chen, L. (2019). Nscaching : simple and efficient negative sampling for knowledge graph embedding. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 614–625. IEEE.

[Zhao and Wang, 2018] Zhao, C. and Wang, Z. (2018). Gogo : An improved algorithm to measure the semantic similarity between gene ontology terms. *Scientific reports*, 8(1) :15107.

[Zheng et al., 2020] Zheng, D., Song, X., Ma, C., Tan, Z., Ye, Z., Dong, J., Xiong, H., Zhang, Z., and Karypis, G. (2020). Dgl-ke : Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 739–748.

[Zhong et al., 2018] Zhong, J., Wang, C., Li, Q., and Li, Q. (2018). A new graph-partitioning algorithm for large-scale knowledge graph. In *International Conference on Advanced Data Mining and Applications*, pages 434–444. Springer.

[Zhu et al., 2022a] Zhu, C., Yang, Z., Xia, X., Li, N., Zhong, F., and Liu, L. (2022a). Multimodal reasoning based on knowledge graph embedding for specific diseases. *Bioinformatics*, 38(8) :2235–2245.

[Zhu et al., 2022b] Zhu, J., Wang, J., Han, W., and Xu, D. (2022b). Neural relational inference to learn long-range allosteric interactions in proteins from molecular dynamics simulations. *Nature Communications*, 13(1) :1661.

[Zhu et al., 2018] Zhu, P., Hu, Q., Hu, Q., Zhang, C., and Feng, Z. (2018). Multi-view label embedding. *Pattern Recognition*, 84 :126–135.

[Zhu et al., 2020] Zhu, Y., Che, C., Jin, B., Zhang, N., Su, C., and Wang, F. (2020). Knowledge-

driven drug repurposing using a comprehensive drug knowledge graph. *Health Informatics Journal*, 26(4) :2737–2750.

[Zhu et al., 2019] Zhu, Z., Xu, S., Tang, J., and Qu, M. (2019). Graphvite : A high-performance cpu-gpu hybrid system for node embedding. In *The World Wide Web Conference*, pages 2494–2504.

[Zhuang et al., 2020] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1) :43–76.

[Zoghlami, 2019] Zoghlami, M. (2019). *Multiple instance learning for sequence data : Application on bacterial ionizing radiation resistance prediction.* Theses, Université Clermont Auvergne [2017-2020] ; Université de Tunis El Manar.

[Zoghlami et al., 2020] Zoghlami, M., Aridhi, S., Maddouri, M., and Nguifo, E. M. (2020). Multiple instance learning for sequence data with across bag dependencies. *Int. J. Mach. Learn. Cybern.*, 11(3) :629–642.