Approche hybride basée sur l'apprentissage automatique pour la réduction de graphes

Clement Aralou, Tobias Rupp Marcel, Samba Ndojh Ndiaye, Mohammed Haddad, Hamida Seba

Approche hybride basée sur l'apprentissage automatique pour la réduction de graphes

Clément Aralou*, Tobias Rupp Marcel *,
Samba Ndojh Ndiaye*, Mohammed Haddad*, Hamida Seba *

* Univ Lyon, UCBL, CNRS, INSA Lyon,
LIRIS, UMR5205, F-69622 Villeurbanne, France
{prenom.nom}@univ-lyon1.fr

Résumé. Récemment, l'arrivée de l'apprentissage profond sur les graphes a permis d'obtenir des résultats fructueux sur plusieurs problèmes d'optimisation sur les graphes. Dans ce papier, nous nous intéressons à la réduction de graphes en utilisant des techniques d'apprentissage automatique. La réduction de graphes a pour objectif d'obtenir des graphes plus simples ou plus petits en agrégeant les noeuds ou les arêtes similaires, sans ou avec perte d'information, ou en sparsifiant le graphe en enlevant des arêtes non significatives pour l'application considérée. Nous proposons une méthode hybride qui sparsifie le graphe dans un premier temps, puis dans un second temps, groupe les nœuds, tout en préservant les distances dans le graphe. Pour cela, nous nous appuyons sur une méthode d'apprentissage automatique par renforcement. Pour valider notre approche, nous effectuons une comparaison avec les méthodes déjà existantes.

1 Introduction

Les graphes, aussi appelés réseaux, sont largement utilisés pour modéliser les interactions entre différentes entités dans de nombreux domaines tels que les réseaux sociaux, les réseaux de communication, les interactions moléculaires, les connexions entre les neurones de nos modèles d'apprentissage automatique, etc. Dans la plupart de ces applications, ces graphes sont complexes et surtout de très grande taille, ce qui rend très difficile leur étude et leur analyse. Pour résoudre ce problème, une des solutions consiste à simplifier le graphe en entrée en réduisant sa taille afin d'obtenir une version plus petite tout en conservant une ou plusieurs propriétés du graphe initial. Les propriétés préservées dépendent de l'application. Par exemple, dans les réseaux sociaux, on peut s'intéresser à une réduction qui préserve les communautés qui existaient dans le graphe initial. Le but de préserver des propriétés du graphe initial dans le graphe réduit est de pouvoir utiliser celui-ci à la place du graphe initial dans l'application sous-jacente. Il existe plusieurs méthodes pour réduire un graphe. Certaines méthodes procèdent par agrégation en regroupant les noeuds similaires : les noeuds qui ont le même voisinage, les noeuds qui portent les mêmes attributs, les noeuds qui appartiennent à la même communauté, etc. Ces méthodes construisent un graphe réduit, un résumé, ou chaque noeud est composé de plusieurs noeuds. D'autres méthodes procèdent par sparsification (dé-densification). Ces

méthodes suppriment des arêtes, et quelques fois des noeuds, qui sont jugées peu pertinentes pour l'application. On dit qu'elles construisent un squelette du graphe initial. La plupart des méthodes agrégatives sont sans perte. Autrement dit, on peut reconstruire le graphe initial à partir de son résumé. Par contre, les méthodes qui sparsifient le graphe ne permettent pas de retrouver le graphe initial.

Avec les succès de l'apprentissage profond dans la résolution de plusieurs problèmes dans divers domaines, il n'est pas surprenant de voir l'arrivée de méthodes d'apprentissage profond sur les graphes pour résoudre des problèmes d'optimisation sur les données de types graphes. Dans ce travail, nous nous intéressons aux méthodes basées sur l'apprentissage profond pour la réduction de graphes en mettant particulièrement l'accent sur la réduction de graphes préservant les distances entre les noeuds du graphe. Autrement dit, dans le graphe réduit obtenu, deux noeuds qui étaient à une distance d l'un de l'autre dans le graphe initial, restent à cette distance, à un facteur additif ou multiplicatif près, dans le graphe réduit. L'objectif de cette réduction préservant les distances est de pouvoir, par exemple approximer une requête de chemins dans le graphe. L'approche proposée, dans ce papier, est une approche hybride qui réalise à la fois une dé-densification du graphe et une agrégation de noeuds. L'étape de dé-densification s'appuie sur l'apprentissage par renforcement pour trouver les meilleures arêtes à supprimer tout en préservant les distances entre les noeuds. L'étape d'agrégation repose sur des propriétés de dominance dans le graphe pour regrouper certains noeuds et réduire encore plus le résumé obtenu. Nous comparons l'approche proposée à plusieurs méthodes de référence afin d'évaluer ses performances.

Le reste du papier est structuré comme suit. Nous commençons par définir les concepts de base dans la section 2. Ensuite, la section 3 résume les travaux de la littérature qui utilisent l'apprentissage automatique pour réduire ou simplifier un graphe. La section 4 présente l'approche proposée ainsi que les concepts associés. La section 5 présente l'étude expérimentale que nous avons menée pour évaluer notre approche. Le papier se termine par une conclusion qui présente une discussion sur les perspectives ouvertes par cette étude.

2 Préliminaires

Graphes et chemins : Un graphe $G = (V, E)$ est un ensemble V de nœuds ou sommets et un ensemble E d'arêtes, tel que $E \subseteq V \times V$. On dit que les sommets u et v sont adjacents s'il existe une arête $\{u, v\} \in E$ qui les relie. Un chemin est une séquence de nœuds telle que deux nœuds successifs dans la séquence sont adjacents dans le graphe. La longueur d'un chemin est le nombre d'arêtes qui le compose. La distance entre deux noeuds u et v, notée $dist(u, v)$, est la longueur du plus court chemin entre ces noeuds. Un graphe est connexe s'il existe un chemin entre chaque paire de noeuds. S'il n'existe pas de chemin reliant u et v, il est souvent considéré que $dist(u, v) = \infty$. Une feuille est un nœud relié à un seul autre nœud, souvent appelé parent. Un cycle est un chemin tel que le sommet de départ est égal au sommet d'arrivée et on ne passe qu'une seule fois par une arête. Un arbre est un graphe connexe sans cycle. Soit un graphe $G = (V, E)$, $T = (V', E')$ est un sous-arbre de G si T est un arbre, $V' \subseteq V$ et $E' \subseteq E$. T est dit k-dominant si, tout nœud qui ne fait pas partie de T, se trouve à au plus k sauts d'un nœud qui en fait partie. Formellement, $\forall v \in V, \exists u \in V^{'}$ tel que $dist(u, v) \leq k$.

Apprentissage Automatique sur les Graphes : L'un des concepts fondamentaux de l'apprentissage automatique sur les graphes réside dans l'apprentissage de représentations pour les nœuds, les arêtes ou pour tout le graphe. L'idée principale consiste à calculer des plongements dans un espace euclidien, sous forme de vecteurs de faible dimension. Il existe de nombreuses méthodes pour calculer les plongements des composants d'un graphe. Dans ce qui suit, nous nous focaliserons sur les *Graph Neural Networks* ou GNN, qui regroupent un ensemble de méthodes utilisant l'apprentissage profond. Plus spécifiquement, nous nous concentrons sur les méthodes utilisant le *passage de messages* qui est une généralisation de la notion de convolution utilisée dans les réseaux de neurones convolutionnels (CNN) (Bruna et al., 2014). Dans ce cadre, chaque élément du graphe met à jour son plongement en agrégeant les plongements de ses voisins. Parmi les GNN, nous avons également des modèles plus spécialisés, notamment les Autoencodeurs. Un Autoencodeur est un modèle qui mappe les données d'entrée vers une représentation de faible dimension (Encodeur) avec la possibilité de les mapper en retour dans la dimension d'origine (Décodeur). Souvent représentés sous la forme d'autoencodeurs, les modèles utilisant du Pooling visent à réduire la taille du signal d'entrée en résumant les informations (Fleuret, 2023).

Apprentissage par renforcement : La principale différence entre l'apprentissage par renforcement et les autres champs de l'apprentissage automatique réside dans le fait que l'objectif n'est pas de découvrir des structures cachées, mais plutôt d'obtenir la récompense maximale à long terme. Dans l'apprentissage par renforcement, l'apprenant (celui qui prend des décisions) est appelé l'agent. Pour parvenir à son but, l'agent doit adopter une approche d'exploration, en prenant des décisions aléatoires pour découvrir quels états entraînent des récompenses positives, puis il passe à l'exploitation de ce qu'il a appris en choisissant les actions qui maximisent la récompense. Les problèmes résolus par l'apprentissage par renforcement sont souvent séquentiels, ce qui signifie qu'un état peut sembler peu prometteur à court terme, mais que les actions suivantes peuvent s'avérer très bénéfiques (Sutton et Barto, 2018).

Processus de décision Markovien : Généralement, le problème de d'apprentissage par renforcement peut être représenté par des processus de décision Markovien (MDP) qui formalisent des problèmes de décision séquentiels. Un MDP est défini par :

- S, un ensemble d'états.
- A, un ensemble d'actions valides.
- $P : S \times A \to \mathcal{P}(S)$ donne la probabilité d'aller dans l'état $s^{'}$ en partant de l'état s et en prenant l'action a.
- $R : S \times A \times S \to \mathbb{R}$ donne la récompense obtenue pour être passé de l'état s à l'état $s^{'}$ en ayant pris l'action a.

Ainsi, Dans ce problème, l'agent interagit avec l'environnement en sélectionnant une action dans l'ensemble **A** en fonction de l'état courant **s**. L'environnement répond en envoyant une récompense et en effectuant une transition vers un nouvel état selon les probabilités dans P. La récompense est donnée par la fonction R, qui capture l'effet immédiat de l'agent sur l'environnement. L'objectif est d'apprendre la politique $\pi : S \to A$ qui maximise la récompense. Nous avons aussi $Q^{\pi}(s, a)$ qui est la récompense espérée en commençant dans l'état **s** et prenant l'action **a**. Q^{π} peut être représentée de manière récursive en utilisant l'équation de

Bellman :

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s'} P(s^{'}|s,a)Q^{\pi}(s^{'},\pi(s^{'})). \quad (1)$$

où $\gamma \in [0,1]$ permet de mettre plus d'importance sur les récompenses plus ou moins loin dans le futur. Dans ce papier, π et Q sont approximées en utilisant notre modèle.

Processus de Markov Décisionnel Partiellement Observable : Dans un POMDP, l'agent n'a qu'une vue partielle de l'environnement. Pour pouvoir prendre des décisions, il doit recueillir des observations afin de déterminer dans quel état il se trouve. Plus formellement, ce processus est défini par :

— (S,A,P,R) un MDP.
— Ω, l'ensemble des observations.
— $\mathcal{O}$, l'ensemble des probabilités associées aux observations.

L'utilisation du POMDP nous permet de travailler sur des parties du graphe initial en considérant des sous-graphes de celui-ci, i.e., des graphes plus petits, ce qui permet de mieux généraliser.

3 Etat de l'art

Les méthodes proposées dans la littérature pour réduire ou simplifier de grands graphes, se focalisent, dans la majorité des cas, sur des graphes issus du monde réel, comme par exemple les réseaux sociaux. Dans ces contextes, il existe souvent une tâche sous-jacente à optimiser comme par exemple la classification des noeuds ou encore la détection d'anomalies. La réduction va tenter de conserver les informations pertinentes nécessaires pour optimiser cette tâche. Les méthodes de réduction peuvent être divisées en deux catégories :

1. L'agrégation : il s'agit d'une technique qui regroupe des nœuds en super-nœuds, avec ou sans perte d'information.
2. La sparsification : l'objectif de cette méthode est de supprimer les arêtes, et éventuellement les noeuds, les moins importants afin d'optimiser la tâche sous-jacente.

Parmi les méthodes d'agrégation, qui utilisent l'apprentissage automatique, nous pouvons citer GOREN (Cai et al., 2021) qui consiste à apprendre à affecter des poids aux arêtes des super-nœuds d'un graphe agrégé précédemment en utilisant notamment des méthodes issues de la théorie spectrale des graphes (*Local variation* (Loukas et Vandergheynst, 2018; Loukas, 2018) ou *Heavy Edge Matching* (Dhillon et al., 2007), etc.). Ceci dans le but de préserver la structure du graphe original. Il existe également des méthodes de réduction de graphes, dites de *Pooling* (Grattarola et al., 2021), qui donnent un nouveau graphe plus petit en agrégeant itérativement les noeuds dont les scores d'importance sont faibles (Liu et al., 2023). La méthode Graph U-Net (Gao et Ji, 2019) est un autoencodeur composé de deux parties : un encodeur pour compresser le graphe et un décodeur pour le reconstruire. L'encodeur sélectionne les noeuds avec le plus d'informations pertinentes, tandis que le décodeur reconstruit le graphe en utilisant notamment le graphe compressé de la couche précédente ainsi que les indices des nœuds supprimés. MIAGE (Ge et al., 2021) tente d'améliorer G-Unet, en définissant une couche supplémentaire pour apprendre une meilleure représentation des nœuds à l'instar des techniques sur l'analyse d'images afin de considérer différents aspects du graphe. Ensuite, elle calcule un

score de similarité entre les noeuds, puis élimine les noeuds avec le score de similarité le plus faible. Pour la reconstruction du graphe, MIAGE utilise les informations des arêtes et le graphe de la couche précédente. Enfin, nous pouvons citer G-CREWE (Qin et al., 2020) qui a pour tâche sous-jacente l'alignement de deux graphes. Elle apprend les représentations des nœuds, puis utilise des heuristiques pour compresser le graphe en créant des super-nœuds. Ensuite, elle utilise les représentations des nœuds pour calculer un alignement entre les nœuds et les super-nœuds des deux graphes considérés dans la tâche d'alignement.

Pour la sparsification de graphes, il existe aussi plusieurs approches dans la littérature. NeuralSparse (Zheng et al., 2020) apprend à compresser un graphe en se basant sur la qualité des solutions de la tâche sous-jacente pour mettre à jour son modèle d'apprentissage. FASTGAT (Srinivasa et al., 2020) représente le graphe comme un circuit électrique puis utilise la résistance électrique pour choisir les arêtes à supprimer. La tâche sous-jacente de cette méthode est d'accélérer les calculs d'attention par la méthode GAT. SparRL (Wickman et al., 2021) est une méthode de sparsification qui utilise l'apprentissage par renforcement pour optimiser une tâche sous-jacente, comme le Page Rank ou les plus courts chemins dans le graphe. La méthode commence par supprimer des arêtes du graphe, ensuite elle calcule un score qui évalue l'impact (positif ou négatif) de cette suppression afin de mettre à jour les poids du modèle en conséquence. Elle utilise notamment les POMDP qui permettent de modéliser des problèmes complexes de prise de décision séquentielle dans un environnement incertain (Arcieri et al., 2023).

4 HyReD : une approche hybride pour une réduction de graphes préservant les distances

Dans cette section, nous présentons une nouvelle approche de réduction de graphes qui a pour objectif de préserver les distances entre les noeuds. Cette approche, appelée HyReD, a la particularité d'allier sparsification et agrégation pour une réduction optimale des graphes en entrée.

Soit $G = (V, E)$ un graphe et $T = (V, E^{'})$ le graphe réduit de G tel que $|E'| \leq |E|$. On dit que les distances sont préservées s'il existe $K \in \mathbb{R}$ tel que pour toute paire de noeuds u et v : $d_T(u, v) \leq K \cdot d_G(u, v)$. On peut ainsi retrouver les distances dans G à partir des distances dans T à un facteur multiplicatif près. La distorsion est le ratio entre les distances dans le graphe réduit et les distances dans le graphe original. Formellement, elle est calculée par la formule suivante :

$$\text{Distorsion}_{\text{G}} = \binom{|V|}{2} \sum_{u,v \in V} \frac{dist_T(u, v)}{dist_G(u, v)}. \tag{2}$$

HyRED est constituée de 2 étapes. La première correspond à la tâche de sparsification et la seconde, à celle de l'agrégation.

Etape 1 - Sparsification : L'objectif de cette étape est de supprimer le maximum d'arêtes tout en minimisant la distorsion. Pour ce faire, nous proposons d'utiliser l'apprentissage par renforcement en adaptant un framework général SparRL (Wickman et al., 2021) à la préservation de la distorsion. Le but est de sparsifier le graphe en supprimant les arêtes ayant l'impact le plus faible sur la distorsion. L'apprentissage par renforcement est utilisé pour choisir les arêtes

à supprimer. Le modèle apprend à sélectionner ces arêtes en exécutant l'algorithme à plusieurs reprises, i.e., un nombre prédéfini d'épisodes. L'algorithme commence par construire un sous-graphe initial $G^{'}$ en supprimant aléatoirement des arêtes, permettant ainsi de travailler dans un environnement partiel. Notre première modification de SPARL consiste à limiter le nombre d'arêtes à supprimer, avant de commencer un épisode, à au plus la moitié du nombre d'arêtes nécessaires pour obtenir un arbre. Ensuite, nous entrons dans la boucle d'apprentissage, où nous avons introduit de l'aléatoire en modifiant les poids des arêtes et en calculant un arbre couvrant minimal. Cela évite la déconnexion du graphe, qui est source d'instabilité pendant l'apprentissage. De plus, le squelette (l'arbre couvrant minimal) choisi sera toujours différent, forçant l'agent à explorer. Ensuite, l'algorithme sélectionne un sous-graphe induit par $|EH|$ arêtes de $G^{'}$ qui ne font pas partie de l'arbre couvrant minimal (ligne 6). Pour des raisons de performances, le modèle prend ses décisions dans ce sous-graphe. Travaillant dans un environnement partiel, l'agent collecte et utilisera les observations : d_{H_t}, η_t et $\mathcal{N}_t$, qui lui permettront à long terme de prendre de meilleures décisions en déterminant le sous-graphe dans lequel il se trouve (lignes 7 et 8). L'agent choisit ensuite quelles arêtes supprimer en utilisant une politique basée sur une probabilité ϵ d'agir aléatoirement. Le reste du temps, il choisit la meilleure arête à supprimer en utilisant les q_values fournis par le modèle. Enfin, pour évaluer la qualité de la suppression, l'algorithme calcule la récompense avec l'équation 2. Cette évaluation permet de déterminer si la suppression de l'arête à été bénéfique. Le modèle enregistre ensuite cette "expérience" dans un tampon (par exemple, une table) (ligne 11). Pour pouvoir apprendre, il échantillonne ensuite des expériences de ce tampon et tente de minimiser l'équation suivante : new_value $\leftarrow$ old_value + step_size $\cdot$ (target $-$ old_value) (ligne 12), représentant l'erreur entre la récompense effective et celle espérée. Cette étape modifie les poids du modèle, permettant ainsi à celui-ci d'apprendre.

Algorithm 1 Etape1 : Sparsification

1: **Entrées :** $G = (V, E)$ un graphe, numEpisodes un entier positif, |EH| une taille de sous-graphe
2: **for** $i = 0$ to numEpisodes **do**
3: Construire $G^{'}$ un environnement partiel en supprimant aléatoirement des arêtes
4: **for** $i = 1$ to $T = (|E^{'}| - (|V| - 1))$ **do**
5: MST $\leftarrow$ un arbre couvrant aléatoire à partir de $G^{'}$
6: $H_t \leftarrow$ un sous graphe induit par |EH| arêtes qui ne font pas partie de MST
7: $\Omega \leftarrow$observations sur le graphe H_t : degré des noeuds d_{H_t}, voisinage à 1 saut $\mathcal{N}_t$, ratio des dégrée η_t
8: $q\_values \leftarrow$ Appel au modèle SparRL pour générer la valeur des arêtes en fonction du graphe
9: Supprimer l'arête en utilisant une stratégie ϵ-greedy à partir des q_values.
10: $r_t \leftarrow$ Récompense($G^{'}$) ;
11: Stocker l'expérience dans un tampon.
12: Échantillionnner des expériences à partir du tampon puis entraîner SparRL($H_t, d_{H_t}, \eta_t, \mathcal{N}_t$)

Etape 2 - Agrégation : L'agrégation s'appuie sur les propriétés de dominance dans le graphe pour compresser un peu plus le graphe réduit obtenu après la première étape. En effet, nous allons nous restreindre à un sous-arbre $k-$dominant. L'intérêt de la compression de notre arbre dans un sous-arbre k-dominant est d'obtenir un squelette du graphe original, tel que tous les nœuds qui ne font pas partie de la solution se trouvent à au plus k sauts d'un nœud qui en fait partie. Pour extraire le sous-arbre dominant, nous allons utiliser l'algorithme 2.

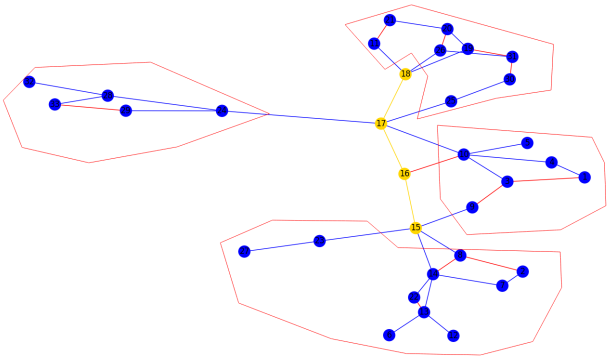

FIG. 1 – Application de HyReD à un graphe. Les arêtes rouges sont celles supprimées durant l'étape de sparsification première phase. Les noeuds de couleur jaune sont les résultats de l'algorithme pour obtenir un sous-graphe 3-dominant. En rouge, nous avons les agrégats de noeuds produits lors de l'agrégation.

Intuitivement, cet algorithme supprime les feuilles de manière récursive tant qu'elles ne sont pas à une distance k d'un nœud déjà supprimé. Pour cela, il construit un tableau associatif qui, pour chaque noeud parent, conserve la distance à laquelle le nœud vient d'être supprimé (voir ligne 7).

Les noeuds de l'arbre $k-$dominant obtenu sont des super-nœuds qui agrègent tous les noeuds qui s'y rattachent et ne font pas partie de l'arbre $k-$dominant. Un exemple est présenté dans la Figure 1.

Algorithm 2 Calcul d'un arbre k-dominant .

1: **Input :** T_0 un arbre, k un entier positif
2: **for** $v \in T_0$ **do**
3: $\quad depth[v] \leftarrow 0$
4: $T \leftarrow T_0$
5: **while** $|T| > 1$ **et** il existe une feuille l telle que $depth[l] < k$ **do**
6: $\quad father[l] \leftarrow$ l'unique voisin de l
7: $\quad depth[father[l]] \leftarrow \max(depth[father[l]], depth[l] + 1)$
8: $\quad$ supprimer l de T
9: **return** T

5 Étude expérimentale

Pour évaluer notre méthode de réduction hybride de graphes, nous avons considéré plusieurs datasets et nous avons comparé ses performances à celles des algorithmes de référence. Nous considérons principalement l'étape de sparsification dans cette évaluation. Les tests ont été effectués sur une machine avec 11 Go de mémoire, une carte graphique RTX 2070 et un i7-4970k pour CPU. Tous les algorithmes ont été exécutés 10 fois, et les résultats présentés,

distorsion et durée d'exécution, sont la moyenne de ces exécutions. Nous avons également calculé la moyenne des déviations standards sur les 10 exécutions. Les seuls hyper-paramètres modifiés par rapport à ceux de la méthode de base SparRl sont le nombre d'épisodes et la taille du sous-graphe |EH|.

Datasets : Nous avons considéré plusieurs modèles de graphes aléatoires et synthétiques, qui tentent de reproduire certaines propriétés de graphes réels. Ces modèles sont définis comme suit :

- Graphes Petit Monde (Watts et Strogatz, 1998) : construit un graphe garantissant que deux noeuds du réseau sont susceptibles d'être reliés par une courte séquence de noeuds intermédiaires. Les graphes "Petit Monde" présentent des propriétés qui se rapprochent des graphes réels. Ils sont robustes, c'est à dire que les nœuds sont fortement connectés.
- Graphes Barabási–Albert (Barabasi et Albert, 1999) : construit un graphe qui obéit à une distribution en loi de puissance pour les degrés des noeuds. Les noeuds ont ainsi une probabilité plus élevée de se connecter à un noeud populaire.
- Graphes réels : Les graphes utilisés font partie du dataset ENZYMES (Borgwardt et al., 2005; Schomburg et al., 2004), qui est un ensemble de données populaire pour les tests d'algorithmes de graphe utilisant l'apprentissage automatique.

Algorithmes de référence : Nous avons comparé notre approche à six autres algorithmes de l'état de l'art qui produisent un résultat similaire au nôtre, i.e., un sous-arbre k-dominant du graphe initial. Ces algorithmes sont comme suit :

- **BFS** : cette méthode s'appuie sur un parcours en largeur du graphe (Breadth First Search) pour calculer un arbre couvrant couche par couche, en visitant à chaque fois les nœuds non encore visités. Ensuite, l'algorithme 2 est appliqué.
- **Décomposition en pétales** : utilise un algorithme de décomposition en pétales (Abraham et Neiman) du graphe pour calculer un arbre couvrant en optimisant la distorsion, puis applique l'algorithme 2.
- **Semi-Aléatoire (S-A)** : crée un arbre couvrant en commençant par un nœud aléatoire, puis à chaque étape, ajoute un nœud qui ne crée pas de boucle jusqu'à obtenir un arbre couvrant. Ensuite, il applique l'algorithme 2.
- **Elagage direct (E-D)** : cette méthode construit directement un arbre dominant. Elle supprime les noeuds tant que cela ne déconnecte pas le graphe et si ce n'est pas le seul noeud à une distance $\leq k$ d'un noeud déjà supprimé.
- **Ensemble k-Dominant** : calcule des sous-graphes k-dominants qui ne sont pas connectés, ensuite il les connecte en utilisant le concept d'ensemble couvrant (set cover) qui pour un ensemble X et une collection de sous ensembles $S = (S_1, S_2, S_3, \ldots, S_n)$ consiste à trouver le minimum de sous-ensembles qui couvrent X. On a ainsi une solution à notre problème en sélectionnant S les sous-ensembles k-dominants.
- **Aléatoire** : construit un arbre en enlevant des arêtes aléatoirement puis applique l'algorithme 2.

Résultats et Discussion : Dans un premier temps, nous avons comparé les résultats de ces algorithmes sur des graphes synthétiques. Nous pouvons constater que notre algorithme obtient de meilleurs résultats dans le cas où nous avons peu d'arêtes. Dans les autres cas, nos

résultats se situent dans la moyenne des autres méthodes. Par exemple, dans le cas des graphes "Petit Monde", nous obtenons de meilleurs résultats, encore une fois pour les graphes avec peu d'arêtes. Nous nous sommes également rendu compte que lorsque l'agent prend des décisions en ayant une vue complète du graphe, nous obtenons de meilleurs résultats. Cependant, dans la méthode originale comme le graphe est représenté par une matrice 2D, il est difficile de généraliser notre observation à des graphes plus grands. Malgré de moins bons résultats sur les plus gros graphes, nous obtenons tout de même une exécution qui est meilleure que toutes les autres (voir les tables 4 et 2), ce qui est encourageant quant à la possibilité d'avoir un algorithme qui converge. Nous obtenons des résultats similaires pour les graphes Barabási–Albert (voir les tables 3 et 4) et les graphes Petit Monde (tables 1 et 2). Pour les graphes réels, nous obtenons aussi des résultats similaires mais comme les graphes sont assez petits, nous obtenons de meilleurs résultats que les méthodes de références (table 6). Enfin, on peut voir que notre principal désavantage est le temps d'apprentissage (voir la table 5) qui est bien plus long que le reste même pour des petits graphes.

Algorithme	50 arêtes †	150 arêtes †	225 arêtes	300 arêtes
BFS	1.930	1.919	**1.847**	1.997
Pétales	1.883	1.910	1.949	1.940
Semi-Aléatoire	2.358	2.748	2.711	2.757
Elagage direct	-	2.245	2.051	2.011
k-Dominant	2.457	1.936	1.860	**1.929**
HyRed	**1.168**	**1.768**	1.944	2.332
Aléatoire	1.328	2.356	2.950	3.019

TAB. 1 – Comparaison des mesures de distorsion pour les graphes synthétiques construits avec l'algorithme de **Watts-Strogatz** (Petit Monde). Les paramètres utilisés sont : probabilité de connexion à une autre arête de 20% et connexion à 6 voisins proches dans le graphe. † indique $|\text{EH}| = |E|$. Nous testons avec 20, 50, 75, et 100 noeuds.

Algorithme	50 arêtes †	150 arêtes †	225 arêtes	300 arêtes
BFS	0.124	0.040	0.106	0.122
Pétales	0.07	0.126	0.150	0.084
S-A	0.138	0.195	0.195	0.217
E-D	-	0.264	0.099	0.065
k-Dominant	**0.0**	**0.0**	**0.0**	**0.0**
HyRed	0.239	0.314	0.420	0.243
Aléatoire	0.156	0.170	0.775	0.476

TAB. 2 – Comparaison de la déviation standard de la distorsion pour les graphes synthétiques construits avec l'algorithme de **Watts-Strogatz** (Petit Monde). Nous testons avec 20, 50, 75, et 100 noeuds.

6 Conclusion

Dans ce papier, nous avons présenté les concepts de base d'une nouvelle méthode de réduction de graphes alliant sparsification et agrégation. Nous nous sommes concentrés principale-

Algorithme	50 arêtes	184 arêtes	184 arêtes †	284 arêtes	284 arêtes †	384 arêtes	384 arêtes †
BFS	1.697	1.864	**1.820**	1.791	1.847	1.820	1.732
Pétales	1.845	1.821	1.792	1.929	1.859	1.880	1.868
S-A	2.500	2.735	2.569	2.668	2.591	2.811	2.708
E-D	-	-	-	-	-	-	2.040
k-Dominant	1.801	**1.696**	1.696	**1.667**	**1.667**	**1.696**	1.696
HyRed	**1.230**	1.699	**1.637**	2.183	1.896	2.369	**1.603**
Aléatoire	1.405	2.333	2.417	2.750	2.906	3.447	3.205

TAB. 3 – Comparaison des mesures de distorsion pour les graphes synthétiques construits avec l'algorithme de **Barabási–Albert**. A chaque ajout de noeud, il est relié à 4 autres noeuds. † indique $|\text{EH}| = |E|$. Nous testons avec 20, 50, 75, et 100 noeuds.

Algorithme	50 arêtes	184 arête	184 arêtes †	284 arêtes	284 arêtes †	384 arêtes	384 arêtes †
BFS	0.054	0.163	0.134	0.118	0.137	0.090	0.065
Pétales	0.136	0.161	0.131	0.068	0.150	0.08	0.219
S-A	0.151	0.277	0.124	0.169	0.394	0.243	0.056
E-D	-	-	-	-	-	-	-
k-Dominant	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
HyRed	0.387	0.441	0.258	0.382	0.140	0.287	0.150
Aléatoire	0.274	-	0.165	0.353	0.855	0.499	0.679

TAB. 4 – Comparaison de la déviation standard de la distorsion pour les graphes synthétiques construits avec l'algorithme de **Barabási–Albert**. Nous testons avec 20, 50, 75, et 100 noeuds.

ment sur l'étape de sparsification pour laquelle nous avons instancié un modèle d'apprentissage par renforcement afin qu'il optimise la distorsion dans le graphe réduit obtenu. La distorsion est notre critère applicatif puisque notre but est d'avoir un graphe réduit qui préserve les distances entre les noeuds. Pour cette étape de sparsification, on peut noter que plus le nombre de sous-graphes possibles augmente, plus il est difficile pour notre agent de prendre de bonnes décisions. En effet, il a plus de chance de se trouver dans un état qu'il n'avait jamais exploré auparavant. En outre, il est encourageant de constater que, pour certaines itérations, notre instance obtient de meilleurs résultats par rapport aux autres méthodes de la littérature.
Ce travail ouvre la voie à plusieurs perspectives : (1) Il serait intéressant de considérer des états de départ plus probables dans lesquels les arêtes supprimées au préalable n'ont pas toujours un impact trop négatif sur la distorsion. (2) Il est également souhaitable d'étendre la liste des observations afin de mieux identifier l'état actuel lors de l'apprentissage. (3) Pour limiter la

Algorithme	50 arêtes	184 arêtes	184 arêtes †	284 arêtes	284 arêtes †	384 arêtes	384 arêtes †
BFS	**0.012**	0.016	0.013	**0.031**	0.033	**0.05**	0.055
Pétales	0.013	0.014	**0.013**	0.034	**0.029**	0.062	0.053
S-A	0.014	0.015	0.014	0.033	0.034	0.054	0.054
k-Dominant	0.016	**0.012**	0.016	0.034	0.031	0.054	**0.051**
HyRed	618	582.64	865.74	1250.86	1922.43	2438.44	2208.74

TAB. 5 – Temps moyen pour des tests pour les graphes construits avec **Barabási–Albert** donné en secondes. Pour HyRED, on considère le temps d'entraînement. En bleu le temps d'entraînement le plus faible.

Algorithme	51 arêtes	84 arêtes	120 arêtes	133 arêtes
BFS	1.597	1.382	1.660	1.703
Pétales	1.443	1.297	1.596	1.643
Semi-Aléatoire	1.631	1.616	1.875	1.829
k-Dominant	1.567	1.377	1.523	1.341
HyRed	**1.013**	**1.202**	**1.405**	**1.222**
Aléatoire	1.367	1.309	1.393	1.507

TAB. 6 – Comparaison de la distorsion pour les graphes du jeux de données **ENZYMES**. Nous testons sur des graphes d'environ 100 noeuds.

durée d'apprentissage, il serait intéressant de travailler sur la fonction de récompense avec un calcul approché de la distorsion, mais aussi en pré-entraînant le modèle au préalable avant de l'affiner sur le graphe courant sur moins d'épisodes.(4) On peut aussi explorer l'imitation learning (Sutton et Barto, 2018), qui permettrait à notre agent d'apprendre en imitant un expert, par exemple un algorithme parmi ceux de référence présentés dans le papier. Cela permettrait à notre agent de prendre de meilleures décisions et de converger plus rapidement (Cappart et al., 2021). (5) Il est également intéressant d'utiliser des Graph Neural Networks (GNN) pour traiter des graphes plus grands. En effet, les GNNs montrent des résultats prometteurs pour la résolution de problèmes combinatoires.

N. B. : Ce travail a reçu un financement du projet ANR COREGRAPHIE ANR-20-CE23-0002 ainsi que du département Informatique de Bourg en Bresse.

Références

Abraham, I. et O. Neiman. Using petal-decompositions to build a low stretch spanning tree. STOC '12, pp. 395–406.

Arcieri, G., C. Hoelzl, O. Schwery, D. Straub, K. G. Papakonstantinou, et E. Chatzi (2023). Pomdp inference and robust solution via deep reinforcement learning : An application to railway optimal maintenance.

Barabasi, A.-L. et R. Albert (1999). Emergence of scaling in random networks. *Science 286*(5439), 509–512.

Borgwardt, K. M., C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, et H.-P. Kriegel (2005). Protein function prediction via graph kernels. *Bioinformatics 21*(suppl_1), i47–i56.

Bruna, J., W. Zaremba, A. Szlam, et Y. LeCun (2014). Spectral networks and locally connected networks on graphs.

Cai, C., D. Wang, et Y. Wang (2021). Graph coarsening with neural networks. *CoRR abs/2102.01350*.

Cappart, Q., D. Chételat, E. B. Khalil, A. Lodi, C. Morris, et P. Velickovic (2021). Combinatorial optimization and reasoning with graph neural networks. *CoRR abs/2102.09544*.

Dhillon, I. S., Y. Guan, et B. Kulis (2007). Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*(11), 1944–1957.

Fleuret, F. (2023). *The Little Book of Deep Learning*.

Gao, H. et S. Ji (2019). Graph u-nets. *CoRR abs/1905.05178*.

Ge, Y., Y. Pang, L. Li, et L. Itti (2021). Graph autoencoder for graph compression and representation learning. In *Neural Compression : From Information Theory to Applications–Workshop@ ICLR 2021*.

Grattarola, D., D. Zambon, F. M. Bianchi, et C. Alippi (2021). Understanding pooling in graph neural networks. *CoRR abs/2110.05292*.

Liu, C., Y. Zhan, J. Wu, C. Li, B. Du, W. Hu, T. Liu, et D. Tao (2023). Graph pooling for graph neural networks : Progress, challenges, and opportunities.

Loukas, A. (2018). Graph reduction by local variation. *CoRR abs/1808.10650*.

Loukas, A. et P. Vandergheynst (2018). Spectrally approximating large graphs with smaller graphs. *CoRR abs/1802.07510*.

Qin, K. K., F. D. Salim, Y. Ren, W. Shao, M. Heimann, et D. Koutra (2020). G-CREWE : Graph CompREssion with embedding for network alignment. ACM.
en

Schomburg, I., A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, et D. Schomburg (2004). BRENDA, the enzyme database : updates and major new developments. *Nucleic Acids Res 32*(Database issue), D431–3.

Srinivasa, R. S., C. Xiao, L. Glass, J. Romberg, et J. Sun (2020). Fast graph attention networks using effective resistance based graph sparsification. *CoRR abs/2006.08796*.

Sutton, R. S. et A. G. Barto (2018). *Reinforcement Learning : An Introduction*. Cambridge, MA, USA : A Bradford Book.

Watts, D. J. et S. H. Strogatz (1998). Collective dynamics of 'small-world' networks. *Nature 393*(6684), 440–442.

Wickman, R., X. Zhang, et W. Li (2021). Sparrl : Graph sparsification via deep reinforcement learning. *CoRR abs/2112.01565*.

Zheng, C., B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, et W. Wang (2020). Robust graph representation learning via neural sparsification. In *Proc. 37th International Conference on Machine Learning*, Volume 119, pp. 11458–11468. PMLR.

Summary

In this paper, we focus on graph reduction using Machine Learning techniques. Recently, the arrival of Deep Learning on graphs led to fruitful results on optimization problems with graphs. The objective of graph reduction is to obtain smaller and simpler graphs, without losing too much information, by grouping similar nodes or edges, or by sparsifying the graph by removing less important edges for the downstream task. We proposed a two steps hybrid method. The first step sparsifies the graph and in the second step, we group the nodes into supernodes while preserving the distances in the graph. Finally, we make a comparison with the existing methods.