Une nouvelle approche pour la détection d'anomalies dans les flux de graphes hétérogènes

Abd Errahmane Kiouche[*,**], Karima Amrouche[*]
Hamida Seba[**], Sofiane Lagraa[***]

[*]Laboratoire de la Communication dans les Systèmes Informatiques(LCSI)
École nationale Supérieure d'Informatique, BP 68M, 16309, Oued-Smar, Alger, Algérie.
http://www.esi.dz
[**]Université de Lyon, CNRS, Université Lyon 1,
LIRIS, UMR5205, F−69622 Lyon, France.
[***]SnT, Université du Luxembourg, L-1855 Luxembourg

Résumé. Nous proposons dans ce travail une nouvelle approche de détection d'anomalies dans un flux de graphes hétérogènes orientés et étiquetés. Notre approche utilise une nouvelle représentation des graphes par des vecteurs. Cette représentation est flexible et permet de mettre à jour les vecteurs de graphes de manière incrémentale à fur et à mesure de l'arrivée de nouvelles arêtes. Elle est applicable à n'importe quel type de graphes et optimise l'espace mémoire utilisé. De plus, elle permet la détection d'anomalies en temps réel.

1 Introduction

La détection d'anomalies est un domaine de recherche très actif traité par plusieurs communautés scientifiques telles que : la sécurité informatique, la médecine, l'industrie et la finance. De façon générale, ce problème consiste à détecter les données qui sont significativement différentes des données bénignes ou normales. De nos jours, les données sont de plus en plus représentées par les graphes car ces derniers ont la faculté de modéliser les interactions complexes de façon simple et intuitive. Un graphe $G = (V, E)$ est un outil de représentation de données formé d'un ensemble de sommets V et d'un ensemble de liens (arêtes) E entre les sommets. Lorsque les données sont représentées par des graphes, le problème de détection d'anomalies revient à repérer les graphes qui sont différents des graphes correspondants aux objets normaux observés par le système. De plus, les graphes en flux (graph stream) sont de plus en plus utilisés. En effet, dans la plupart des applications de surveillance en temps réel, la structure complète des graphes n'est pas connue, car les graphes grandissent et évoluent au fil du temps. De même, lorsque les graphes sont trop volumineux pour être chargés entièrement en mémoire centrale, les traiter dans le modèle de flux de données où le flux est en général une séquence d'arêtes est une nécessité. La détection d'anomalies dans un flux d'arêtes pose plusieurs défis comme le traitement incrémental des arêtes, la gestion de l'espace mémoire occupé par le flux et la détection des anomalies en temps réel.

Dans ce travail, nous nous intéressons au problème de la détection d'anomalies dans un flux de graphes hétérogènes et étiquetés. Notre application principale est la sécurité des systèmes

informatiques. Chaque graphe dans le flux représente une fenêtre d'une activité particulière du système (accès mémoire, authentification, etc.). Ceci explique l'hétérogénéité du flux.

Considérons un flux d'arêtes provenant de différents graphes hétérogènes orientés et étiquetés. Chaque arête du flux est représentée par un 6-upplet <sommet source, l_s, sommet destination, l_d, l_e, $id\ graphe >$ où l_s, l_d et l_e représentent les étiquettes du sommet source, du sommet destination et de l'arête respectivement. Le flux d'arêtes forme des graphes dynamiques qui évoluent au fil du temps. Les arêtes qui partagent le même $id\ graphe$ appartiennent au même graphe. De plus les arêtes qui proviennent de graphes différents peuvent être entrelacées et donc plusieurs graphes peuvent évoluer simultanément. La problématique considérée ici est de détecter, dans ce flux, les graphes anormaux à n'importe quel moment t.

Un graphe anormal est défini comme étant un graphe qui est significativement différent des graphes bénins ou normaux connus par le système. Ainsi, la détection d'anomalies dans ce flux d'arêtes peut être vue comme un problème de comparaison/classification de graphes : au fur et à mesure que les graphes évoluent avec l'arrivée de nouvelles arêtes, on les re-classifie en normaux ou anormaux selon leur similitude avec des graphes d'entraînement qui représentent un comportement normal du système. Les deux problématiques sous-jacentes sont donc : (1) comment calculer la similarité entre les graphes ? et (2) comment les classifier ?

Comparer deux graphes est un problème complexe dont les solutions sont généralement exponentielles (Bunke et Allerman, 1983). Pour obtenir des approches de comparaison polynômiales, la méthode la plus utilisée est de décomposer les deux graphes à comparer en sous-structures plus simples et de comparer les sous-structures obtenues (Riesen et al., 2015). Il existe plusieurs méthodes de calcul de similarité entre les graphes en utilisant leurs sous-structures. Les méthodes basées sur les noyaux de graphes (Shervashidze et al., 2011, 2009) sont les plus rapides mais ne sont pas applicables dans le cas de graphes dynamiques car elles pré-calculent un espace fixe de sous-structures pour représenter les graphes alors que dans un flux la structure complète des graphes change au fil du temps. Les méthodes basées sur la distance d'édition de graphes (GED pour Graph Edit Distance) ne peuvent être utiles non plus car on doit recalculer la GED à chaque arrivée d'une nouvelle arête ce qui est très coûteux. Rappelons que la GED définit la similarité entre deux graphes par la séquence minimale d'opérations d'éditions (i.e., insertions ou suppressions de nœuds ou d'arêtes) nécessaires pour transformer un graphe en l'autre (Sanfeliu et Fu, 1983). Les approximations les plus rapides de la GED sont de complexité polynomiale (Fischer et al., 2017; Riesen et al., 2015).
La phase de classification est une problématique importante. En effet, l'inconvénient principal de la représentation par graphes est le manque de méthodes appropriées pour la classification et le clustering dans l'espace des graphes. Cela est dû principalement au fait que certaines opérations de base nécessaires dans la classification ne sont pas disponibles pour les graphes (Riesen et al., 2007). Une technique pour pallier ce problème consiste à transformer les graphes en vecteurs pour pouvoir appliquer les algorithmes classiques de classification, on parle dans ce cas de plongement de graphes (graph embedding). D'une manière générale, le plongement de graphes consiste à faire associer à chaque graphe un point dans un espace vectoriel, de telle sorte que les points qui sont associés aux graphes similaires soient proches (Foggia et al., 2014). Une des approches de plongement de graphes qui a prouvé son efficacité pour la classification consiste à représenter un graphe G par un vecteur $E_G = (d_1, ..., d_M)$ contenant les distances d'édition entre le graphe G et M graphes prototypes sélectionnés dans l'ensemble des graphes d'entraînement (Riesen et Bunke, 2009). Cependant, cette méthode est inefficace

dans le cas de flux d'arêtes, car le coût nécessaire pour mettre à jour la GED de manière incrémentale (càd à chaque arrivée d'une nouvelle arête) est de complexité quadratique $\mathcal{O}(n^2)$ où n représente le nombre de sommets dans le graphe (Mills-Tettey et al., 2007; Toroslu et Üçoluk, 2007).

Nous proposons dans cet article une nouvelle approche pour la détection d'anomalies en temps réel dans un flux de graphes hétérogènes et étiquetés tout en prenant en compte les défis de la gestion de manière incrémentale des arêtes lors de la comparaison des graphes, et de la limitation d'espace mémoire.

La suite de cet article est organisée en 4 sections : La deuxième section présente l'état de l'art. La section 3 est consacrée à la description de l'approche proposée. La section 4 présente la complexité de la méthode en termes de temps et d'espace, ainsi que les résultats que nous avons obtenus par expérimentations. Enfin, la dernière section conclut l'article en présentant quelques perspectives.

2 État de l'art

Le problème de la détection d'anomalies dans les graphes a fait l'objet de plusieurs travaux (Akoglu et al., 2015; Ranshous et al., 2015). Cependant, la plupart des approches existantes ne portent pas sur la détection de graphes anormaux mais plutôt sur la détection d'objets anormaux dans les graphes tels que les sommets anormaux (Akoglu et al., 2010; Papalexakis et al., 2012), les sous-graphes anormaux (Noble et Cook, 2003) ou les communautés et les événements anormaux (Sun et al., 2010; Aggarwal et Subbian, 2012). Parmi les méthodes proposées pour la detection de graphes anormaux dans un flux, nous pouvons citer Classy (Kostakis, 2014), une approche distribuée pour la détection des programmes malveillants dans un flux de graphes orientés et étiquetés représentant des appels de fonctions (call graphs). Classy compare deux graphes avec une approximation de la GED qui utilise le recuit simulé. De plus, afin d'accélérer le processus de la classification des nouveaux graphes entrants, elle utilise une borne inférieure de la GED de complexité temporelle $\mathcal{O}(n)$ où n est le nombre de sommets dans le graphe. Cependant, Classy est conçue pour des flux de graphes entiers et non pour un flux d'arêtes. Spotlight (Eswaran et al., 2018) est un approche basée sur le sketching des graphes pour la détection des graphes anormaux dans un flux d'arêtes de graphes bipartis, orientés et pondérés. Dans cette méthode, l'anomalie est définie comme la disparition ou l'apparition soudaine d'un sous-graphe dense dans un graphe. Le point fort de cette approche est qu'elle arrive à représenter chaque graphe par un vecteur de taille fixe et réduite appelé sketch. Chaque dimension du sketch représente la somme des poids des arêtes d'une région (sous-graphe) du graphe. Les graphes anormaux peuvent être détectés en repérant les sketchs les plus éloignés des sketchs normaux dans l'espace vectoriel. Spotlight ne traite que les graphes bipartis simples et ne détecte qu'un type spécifique d'anomalies (l'apparition ou disparition soudaine d'un sous-graphe dense). StreamSpot (Manzoor et al., 2016) est applicable directement à notre problématique. StreamSpot decompose un graphe en $k-$shingles qui sont des arbres de profondeur k et utilise une extension de la similarité cosinus pour les comparisons. Cependant, l'inconvénient principal de cette similarité est qu'elle ne prend en compte que le nombre des sous-structures communes entre les deux graphes à comparer et ne fait aucune comparaison entre les sous-structures. Cela rend cette similarité non précise dans le cas où les graphes sont très denses ou dans le cas où les sous-structures sont grandes. Pour remédier à ce problème

les auteurs ont divisé les sous-structures en petits morceaux de taille fixe. Cependant, le choix de la taille C des morceaux influe significativement sur la précision de la similarité. En effet, un petit C rend la plupart des paires de graphes similaires, tandis qu'un grand C rend les paires de graphes plus dissemblables. L'inconvénient de cette solution est donc le re-calibrage du paramètre C à chaque arrivée d'un nouveau type de graphes bénins. De plus, pour être incrémentale, l'approche sauvegarde dans un cache de taille limitée les arêtes des graphes. Cependant, lorsque ce cache est plein, StreamSpot supprime les anciennes arêtes pour stocker les nouvelles qui arrivent. Par conséquent, une partie de chaque graphe sera perdue, ce qui influe sur le la précision de la détection.

3 Approche proposée

Dans cette section, nous allons présenter notre approche en commençant par décrire la représentation que nous proposons pour les graphes, ensuite nous décrirons le processus de détection d'anomalies basé sur cette représentation.

3.1 Représentation des Graphes

3.1.1 La décomposition des graphes en sous-structures

Nous décomposons chaque graphe en un ensemble de sous-structures locales appelées branches. Notre décomposition est une extension de la décomposition en branches de Zheng et al. (2013) au cas des graphes orientés. Chaque branche est constituée d'un sommet r et des arêtes dont r est l'origine ou l'extrémité. La figure 1 illustre cette decomposition. Chaque

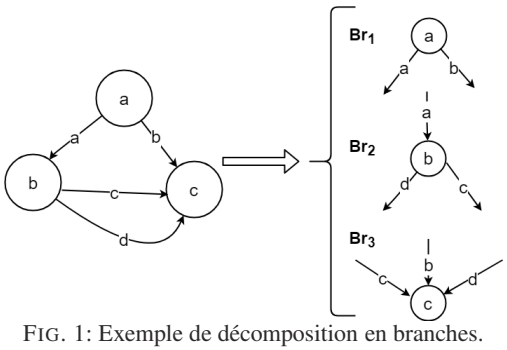


FIG. 1: Exemple de décomposition en branches.

branche est représentée par un couple (r, ES) où r est l'étiquette du nœud racine et ES est un vecteur contenant les arêtes voisines du nœud r. Nous utilisons la structure de données introduite dans (Lopresti et Wilfong, 2003) et appelée "structure d'arêtes" pour les vecteurs ES. Supposons qu'il existe α étiquettes d'arêtes différentes $l_1, ..., l_\alpha$, le vecteur ES de la branche associée au sommet r contient 2α entiers non négatifs, $(x_1, ..., x_\alpha, y_1, ..., y_\alpha)$, tel que x_i est le nombre d'arêtes sortantes de r marquées par l_i et y_j est le nombre d'arêtes vers r marquées par l_j. Pour simplifier, on scinde le vecteur ES en 2 vecteurs ES_{OUT} et ES_{IN} contenant

la structure des arêtes sortantes de r et celles entrantes vers r respectivement. Plus formellement, $ES_{OUT} = (x_1, ..., x_\alpha)$ et $ES_{IN} = (y_1, ..., y_\alpha)$. La table 1 illustre la représentation des branches de l'exemple de la figure 1.

Branches	ES_{OUT}				ES_{IN}			
	a	b	c	d	a	b	c	d
BR_1	1	1	0	0	0	0	0	0
BR_2	0	0	1	1	1	0	0	0
BR_3	0	0	0	0	0	1	1	1

TAB. 1: Exemple de représentation de branches.

3.1.2 Calcul de similarité entre les branches

Nous proposons d'utiliser la GED pour calculer la similarité entre deux branches. Ce choix est justifié par le fait que cette métrique nous permet de traiter n'importe quel type de graphe (i.e., orienté ou pas, simple ou multigraphe et étiqueté ou non étiqueté). La distance d'édition entre deux branches est définie comme suit :

Definition 1 *(Zheng et al., 2013) Soient $Br_1 = (r_1, ES_1)$ et $Br_2 = (r_2, ES_2)$ deux branches, la distance d'édition entre les deux branches est :*

$$BED(Br_1, Br_2) = T(r_1, r_2) + \Gamma(ES_{1(IN)}, ES_{2(IN)}) + \Gamma(ES_{1(OUT)}, ES_{2(OUT)}) \quad (1)$$

avec $\Gamma(A, B) = \max(|A|, |B|) - \sum_{i=1}^{\alpha} \min(A_i, B_i)$*, et*

$$T(r_1, r_2) = \begin{cases} 1 & si\ r_1 = r_2 \\ 0 & sinon \end{cases} \quad (2)$$

Nous calculons la similarité entre deux branches par :

$$Sim(Br_1, Br_2) = 1 - BED_{nrm}(Br_1, Br_2) \quad (3)$$

où $BED_{nrm}(Br_1, Br_2)$ est la distance d'édition normalisée entre les deux branches définie comme suit :

$$BED_{nrm}(Br_1, Br_2) = \frac{BED(Br_1, Br_2)}{1 + \max(|ES_{1(Out)}|, |ES_{2(OUT)}|) + \max(|ES_{1(IN)}|, |ES_{2(IN)}|)} \quad (4)$$

3.1.3 Sélection des branches prototypes

Nous utilisons la stratégie du prototype couvrant en tenant compte des classes indépendantes pour la sélection des branches prototypes, (En anglais SPS-C « Spanning Prototype Class-wise ») (Riesen et Bunke, 2009). Cette stratégie prend en compte toutes les distances par rapport aux prototypes sélectionnés auparavant. La première branche prototype est la médiane du classe. Chaque branche prototype supplémentaire sélectionnée par notre stratégie SPS-C représente la branche la plus éloignée des branches prototypes déjà sélectionnées. Cette

stratégie prend en compte toutes les distances par rapport aux prototypes déjà sélectionnés et tente de couvrir l'ensemble des branches bénignes le plus uniformément possible (Riesen et Bunke, 2009).

Soient M le nombre total de branches à sélectionner et K le nombre des différentes classes de graphes normaux, la technique consiste à choisir $\frac{M}{K}$ branches prototypes de chaque classe c comme suit :

$$P_i = \begin{cases} mediane(c) & si\ i = 1 \\ P_{i-1} \cup \{p_i\} & si\ 1 < i < \frac{M}{K} \end{cases} \tag{5}$$

avec $p_i = \underset{br \in C \setminus P_{i-1}}{\arg\max}\ \min_{p \in P_{i-1}} BED(br, p)$

3.1.4 Plongement de graphes par sous-structures pondérées

Soit $P = \{sp_1, ..., sp_M\}$ un ensemble contenant M branches prototypes extraites de graphes normaux dans la phase d'entraînement. Pour chaque graphe G, nous maintenons en mémoire une matrice M_G où $BED(i, j)$ représente la distance d'édition entre la i-ème branche du graphe G et la j-ème branche prototype.

$$M_G = \begin{pmatrix} BED(1,1) & \dots & BED(1,M) \\ \vdots & \ddots & \vdots \\ BED(N,1) & \dots & BED(N,M) \end{pmatrix} \tag{6}$$

Nous représentons un graphe G par le vecteur E_G de M composantes, où la i-ème composante représente la somme des similarités pondérées entre les branches de G et la i-ème branche prototype (i.e., $E_G = (e_1, ..., e_M)$ tel que $e_j = \sum_{i=1}^{N} Sim(i, j)w_i$). Le terme $w_i = \frac{|Br_i|}{|G|}$ représente le poids de la branche d'index i. C'est le rapport entre le nombre d'arêtes dans la branche Br_i et le nombre d'arêtes dans le graphe. Le terme $Sim(i, j)$ représente la similarité entre les branches i et j calculée par la formule 3. Le produit $Sim(i, j)w_i$ représente l'impact de la branche i par rapport au prototype j.

Nous calculons la distance entre deux graphes G_1 et G_2 en utilisant la distance euclidienne (norme $L2$) entre leurs deux vecteurs caractéristiques E_{G_1} et E_{G_2} : $d(G_1, G_2) = L_2(E_{G_1}, E_{G_2})$. Il est clair que si deux graphes G_1 et G_2 contiennent des branches similaires, la distance entre eux $d(G_1, G_2)$ sera petite et vice versa. De plus, il est évident qu'en utilisant ce plongement, les graphes anormaux seront les plus éloignés des autres graphes car les branches qu'ils contiennent sont les moins similaires aux branches prototypes.

3.2 Détection d'anomalies

La phase d'entraînement : Avant d'entamer la détection d'anomalies, le système doit être entraîné pour qu'il reconnaisse les modèles de graphes normaux existants. Dans la phase d'entraînement nous avons repris l'algorithme de clustering utilisé dans la méthode StreamSpot (Manzoor et al., 2016), qui consiste à regrouper les graphes d'entraînement dans k clusters en utilisant l'algorithme k-Medoid, le paramètre k est choisi de telle sorte qu'il maximise le coefficient silhouette (Rousseeuw, 1987). Cette technique permet de bien séparer les clusters des graphes les uns des autres. Un seuil d'anomalie est ensuite attribué à chaque cluster en utilisant l'inégalité de Cantelli (Grimmett et Stirzaker, 2001). Ce seuil est fixé à 3 fois l'écart type des

distances plus la distance moyenne entre les graphes du cluster et le graphe médoïde. Ensuite, nous calculons le centroïde de chaque cluster qui est la moyenne des vecteurs caractéristiques des graphes du cluster.

La détection d'anomalies en temps réel : à l'arrivée d'une nouvelle arête e du graphe G sortante du sommet v_i et dirigée vers le sommet v_j et étiquetée l_e, les branches Br_i et Br_j changent et par conséquent le vecteur E_G doit être mis à jour. Deux situations sont alors possibles : soit les branches Br_i et Br_j apparaissent pour la première fois, soit elles existent déjà. Dans le premier cas, nous calculons les distances d'édition entre les deux branches et les branches prototypes, puis nous rajoutons l'impact des nouvelles branches au vecteur E_G. Dans le deuxième cas, nous mettons à jour le vecteur E_G en soustrayant les anciens impacts des deux branches et ensuite en rajoutant leurs nouveaux impacts. La complexité de la mise à jour du vecteur E_G est de $\mathcal{O}(M)$. Notons que dans les deux cas, on met à jour les deux branches. Cette opération est de complexité constante $\mathcal{O}(cste)$. Concernant, le calcul des nouveaux impacts, il revient à mettre à jour les distances d'édition entre les branches Br_i et Br_j et les M branches prototypes. Soient d_{im} et d_{jm} les distances d'édition entre la m-ème branche prototype et les branches Br_i et Br_j respectivement avant l'arrivée de l'arête e, les nouvelles distances d'_{im} et d'_{jm} sont calculées comme suit :

$$d'_{im} = \begin{cases} d_{im} + 1 & Si\ (|ES_{i(OUT)}| + 1 > |ES_{m(OUT)}|)\ et \\ & (ES_{i(OUT)}[l_r] + 1 > ES_{m(OUT)}[l_r]) \\ d_{im} - 1 & Si(|ES_{i(OUT)}| + 1 \leq |ES_{m(OUT)}|)\ et \\ & (ES_{i(OUT)}[l_r] + 1 \leq ES_{m(OUT)}[l_r]) \\ d_{im} & Sinon \end{cases}$$

$$d'_{jm} = \begin{cases} d_{jm} + 1 & Si\ (|ES_{j(IN)}| + 1 > |ES_{m(IN)}|)\ et \\ & (ES_{j(IN)}[l_r] + 1 > ES_{m(IN)}[l_r]) \\ d_{jm} - 1 & Si\ (|ES_{j(IN)}| + 1 \leq |ES_{m(IN)}|)\ et \\ & (ES_{ji(IN)}[l_r] + 1 \leq ES_{m(IN)}[l_r]) \\ d_{jm} & Sinon \end{cases}$$

4 Évaluation

4.1 Étude de complexité

— **Complexité spatiale** : pour chaque graphe G, l'approche consomme un espace mémoire de l'ordre de $\mathcal{O}\left(|V| * (M + 2\alpha)\right)$ car l'espace occupé par les branches du graphe est $\mathcal{O}\left(|V| * (2\alpha)\right)$, l'espace occupé par la matrice M_G est $\mathcal{O}\left(|V| * M\right)$, et l'espace occupé par la vecteur E_G est $\mathcal{O}\left(M\right)$.

— **Complexité temporelle :** à l'arrivée d'une nouvelle arête, le temps nécessaire pour la classification du graphe est de $\mathcal{O}\left(M * k\right)$ car la mise à jour des branches est $\mathcal{O}\left(cste\right)$, la mise à jour des distances d'édition est $O\left(M\right)$, la mise à jour du vecteur E_G est $\mathcal{O}\left(M\right)$, et la classification est $\mathcal{O}\left(k * M\right)$.

4.2 Expérimentation

Nous avons testé notre approche sur une configuration Intel I7 8700K @ 3.7 GHz avec 32Go RAM, en utilisant le dataset sbustreamspot-data[1] crée par Manzoor et al. (2016). Ce dataset consiste en un flux de graphes représentant des activités et des connexions dans un système informatique dans le but de détecter des cyber-attaques. Il contient un scénario d'activités malicieuses et 5 scénarios d'activités bénignes. Les activités bénignes représentent des navigations internet normales, telles que : la navigation sur YouTube, téléchargement de fichiers, navigation sur des sites d'information, consultation d'emails et des jeux vidéo en ligne. Le scénario d'attaque consiste en un téléchargement à la dérobée déclenché par une visite d'une URL malveillante. Les caractéristiques des 6 types de graphes sont présentées dans le tableau 2. Les graphes normaux ont été regroupés en 3 sous-ensembles de données :

— ALL regroupe tous les graphes d'activités bénignes .
— YDC regroupe les activités de : navigation YouTube, téléchargement et navigation sur des sites d'information.
— GFC regroupe les activités de : consultation du courriel, les jeux vidéo et navigation sur des sites d'information.

Activité	# graphes	Moyenne $\lvert V \rvert$	Moyenne $\lvert E \rvert$
Navigation YouTube	100	8291.79	113228
Téléchargement	100	6827.3	37382
Consultation des infos	100	8637.34	112957
Attaque	100	8890.8	28423
Consultation d'émail	100	8831.46	310813
Jeux en ligne	100	8990.09	294903

TAB. 2: Les caractéristiques des graphes du corpus

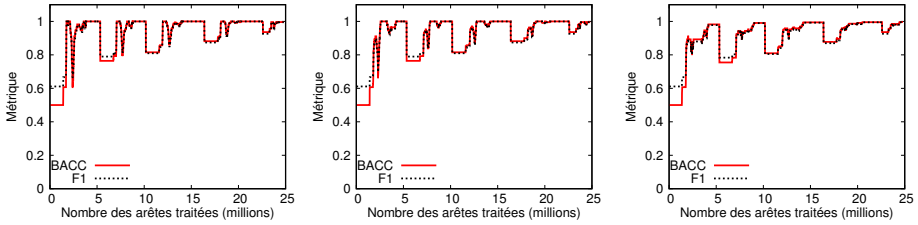
Nous avons testé notre approche en utilisant différents taux d'entraînement $\tau = \{25\%, 50\%, 75\%\}$ sur les 3 sous-ensembles de données ALL, YDC et GFC. De plus, afin d'observer le comportement de notre approche à l'arrivée des nouveaux graphes non vus au préalable, nous avons contrôlé le nombre de graphes qui arrivent et grandissent simultanément en créant des groupes aléatoires de P graphes de tests. Dans toutes les expérimentations, les métriques de performances sont calculées périodiquement à chaque 10000 arêtes. Nous avons utilisé les métriques suivantes comme mesures de performance :

— F1 score : est la moyenne harmonique de la précision et le rappel. Il combine à la fois la précision et le rappel.
— BACC (Balanced Accuracy) : l'exactitude équilibrée est la moyenne arithmétique des taux des vrais positifs et des vrais négatifs.

Étude paramétrique : Cette première série de tests a pour objectif le calibrage du nombre de branches prototypes M. Pour cela, nous avons varié le paramètre $M = \{25, 50, 75\}$ afin d'étudier son impact sur la méthode. Nous avons lancé les 3 exécutions sur le sous-ensemble de données ALL en utilisant $\tau = 75\%$ de graphes bénins dans la phase d'entraînement. Les 25% restants de graphes bénins avec tous les graphes d'attaques constituent l'ensemble des graphes de test qui sont ensuite partitionnés en groupes de $P = 50$ graphes. La figure 2 montre la performance de notre méthode sur différentes valeurs de M. Nous remarquons que les trois

1. https :// github.com/sbustreamspot/sbustreamspot-data

graphiques sont similaires avec des creux périodiques dans chacun d'eux. Chaque creux correspond à l'arrivée d'un nouveau groupe de graphes. Ces baisses de performance sont justifiées par le fait qu'au début de chaque période, seule une petite partie de la structure globale des nouveaux graphes est connue. Les performances se redressent au fur et à mesure que les graphes commencent à grandir pour atteindre un taux de BACC et un score F1 de 99% à la fin de chaque période. Nous remarquons également que les performances pour $M = 25$ sont légèrement meilleures que $M = 50$ et $M = 100$. La table 3 présente la moyenne et l'écart type de chaque mesure de performance, ainsi que le nombre d'arêtes traitées par seconde pour chaque valeur du paramètre M. Nous remarquons que les performances de détection sont presque identiques pour les 3 valeurs de M. Cependant, les durées d'exécution ne le sont pas. En effet, il est évident que plus le nombre M est petit, plus le nombre d'arêtes traitées par seconde est grand. Nous déduisons de la figure 2 et du tableau 3 que la meilleure valeur de M est 25.



(a) $M = 25$ (b) $M = 50$ (c) $M = 100$

FIG. 2: Effet du paramètre M sur les performances de la méthode

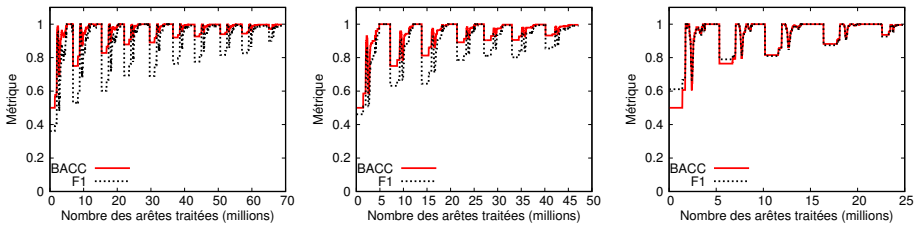
M	Moy BACC	Std BACC	Moy F1 score	Std F1 score	#arêtes/sec
25	0.916	0.132	0.922	0.103	97833
50	0.912	0.132	0.918	0.109	49818
100	0.901	0.126	0.905	0.102	25033

TAB. 3: Performances pour différentes valeurs du paramètre M

Résultats : Les figures 3,4,5 et la table 4 montrent les performances de notre approche sur les trois sous-ensembles de données. Pour les jeux de données ALL et GFC (figure 3), les performances de notre approche sont presque idéales pour les trois taux d'entraînement. Notre méthode atteint 99% du BACC et 99% du score F1 dès que les nouveaux graphes commencent à grandir. Concernant le dataset YDC (figure 4), nos résultats sont idéaux pour les taux d'entraînement 75% et 50% (BACC=100% et F1=100%) et très satisfaisants pour $\tau = 25\%$ (BACC et F1 score >90%). Concernant la vitesse du traitement, notre méthode est rapide et arrive à traiter plus $9 * 10^4$ arêtes par seconde en moyenne.

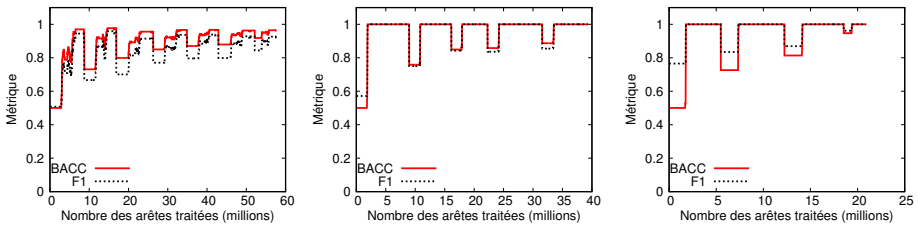
5 Conclusion et perspectives

Dans cet article, nous avons présenté une nouvelle représentation de graphes par des vecteurs pour la détection d'anomalies dans un flux de graphes hétérogènes. Cette representation



(a) $\tau = 25\%$ (b) $\tau = 50\%$ (c) $\tau = 75\%$

FIG. 3: Performances de la méthode sur le jeu de données ALL



(a) $\tau = 25\%$ (b) $\tau = 50\%$ (c) $\tau = 75\%$

FIG. 4: Performances de la méthode sur le jeu de données YDC

permet la mise à jour rapide des vecteurs de graphes et consomme un espace mémoire limité par graphe. De plus, elle peut être appliquée à n'importe quel type de graphes. Les résultats obtenus montrent l'efficacité de notre approche, qui atteint une précision supérieure à 95% en traitant plus de 9×10^4 arêtes par seconde. Comme perspectives, nous pensons pouvoir améliorer notre approche en utilisant d'autres stratégies de sélection des prototypes. Nous envisageons également de comparer notre approche avec les travaux existants et l'étendre à d'autres domaines d'applications comme l'analyse des réseaux sociaux et la biochimie. Une implémentation sur StremSpark serait également intéressante.

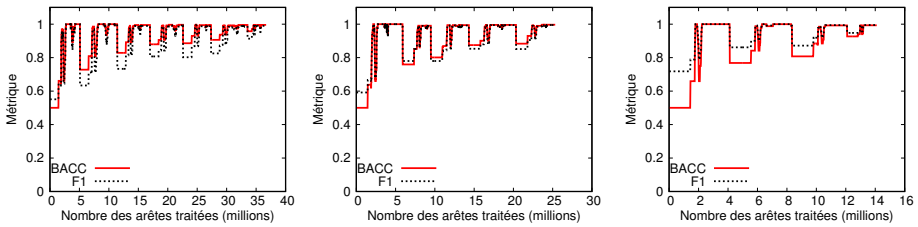
N.B. : Ce travail a été réalisé grace aux séjours scientifiques financés par PHC TASSILI 17MDU984.

Références

Aggarwal, C. C. et K. Subbian (2012). Event detection in social streams. In *Proceedings of the 2012 SIAM international conference on data mining*, pp. 624–635.

Akoglu, L., M. McGlohon, et C. Faloutsos (2010). oddball : Spotting anomalies in weighted graphs. In M. J. Zaki, J. X. Yu, B. Ravindran, et V. Pudi (Eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 410–421. Springer Berlin Heidelberg.

Akoglu, L., H. Tong, et D. Koutra (2015). Graph based anomaly detection and description : A survey. *Data Min. Knowl. Discov. 29*(3), 626–688.



(a) $\tau = 25\%$ (b) $\tau = 50\%$ (c) $\tau = 75\%$

FIG. 5: Performances de la méthode sur le jeu de données GFC

	τ	BACC	F1 Score	#aretes/(sec)
ALL	25%	0.997	0.990	95837
	50%	0.996	0.990	92867
	75%	0.996	0.995	97833
GFC	25%	0.964	0.926	109153
	50%	1.000	1.000	88573
	75%	1.000	1.000	97013
YDC	25%	0.996	0.990	97190
	50%	0.997	0.995	97114
	75%	0.993	0.995	98561

TAB. 4: Performances de notre approche à la fin du flux

Bunke, H. et G. Allerman (1983). Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters - PRL 1*(4), 245–253.

Eswaran, D., C. Faloutsos, S. Guha, et N. Mishra (2018). Spotlight : Detecting anomalies in streaming graphs. In *24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1378–1386.

Fischer, A., K. Riesen, et H. Bunke (2017). Improved quadratic time approximation of graph edit distance by combining hausdorff matching and greedy assignment. *Pattern Recognition Letters 87*, 55 – 62. Advances in Graph-based Pattern Recognition.

Foggia, P., G. Percannella, et M. Vento (2014). Graph matching and learning in pattern recognition in the last 10 years. *International Journal of Pattern Recognition and Artificial Intelligence 28*(01).

Grimmett, G. et D. Stirzaker (2001). *Probability and random processes*. Oxford university press.

Kostakis, O. (2014). Classy : fast clustering streams of call-graphs. *Data Mining and Knowledge Discovery 28*(5), 1554–1585.

Lopresti, D. et G. Wilfong (2003). A fast technique for comparing graph representations with applications to performance evaluation. *Document Analysis and Recognition 6*(4), 219–229.

Manzoor, E., S. M. Milajerdi, et L. Akoglu (2016). Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1035–1044.

Mills-Tettey, G. A., A. Stentz, et M. B. Dias (2007). The dynamic hungarian algorithm for the assignment problem with changing costs. Technical Report CMU-RI-TR-07-27, Robotics

Institute, Carnegie Mellon University.

Noble, C. C. et D. J. Cook (2003). Graph-based anomaly detection. In *9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 631–636.

Papalexakis, E. E., C. Faloutsos, et N. D. Sidiropoulos (2012). Parcube : Sparse parallelizable tensor decompositions. In P. A. Flach, T. De Bie, et N. Cristianini (Eds.), *Machine Learning and Knowledge Discovery in Databases*, pp. 521–536. Springer Berlin Heidelberg.

Ranshous, S., S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, et N. F. Samatova (2015). Anomaly detection in dynamic networks : a survey. *Computational Statistics 7*(3), 223–247.

Riesen, K. et H. Bunke (2009). Graph classification based on vector space embedding. *International Journal of Pattern Recognition and Artificial Intelligence 23*(06), 1053–1081.

Riesen, K., M. Ferrer, A. Fischer, et H. Bunke (2015). Approximation of graph edit distance in quadratic time. In C.-L. Liu, B. Luo, W. G. Kropatsch, et J. Cheng (Eds.), *Graph-Based Representations in Pattern Recognition*, Cham, pp. 3–12. Springer International Publishing.

Riesen, K., M. Neuhaus, et H. Bunke (2007). Graph embedding in vector spaces by means of prototype selection. In F. Escolano et M. Vento (Eds.), *Graph-Based Representations in Pattern Recognition*, Berlin, Heidelberg, pp. 383–393. Springer Berlin Heidelberg.

Rousseeuw, P. J. (1987). Silhouettes : A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics 20*, 53 – 65.

Sanfeliu, A. et K. Fu (1983). A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. on Syst., Man, and Cybernetics (Part B) 13*(3), 353–363.

Shervashidze, N., P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, et K. M. Borgwardt (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research 12*(Sep), 2539–2561.

Shervashidze, N., S. Vishwanathan, T. Petri, K. Mehlhorn, et K. Borgwardt (2009). Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pp. 488–495.

Sun, H., J. Huang, J. Han, H. Deng, P. Zhao, et B. Feng (2010). gskeletonclu : Density-based network clustering via structure-connected tree division or agglomeration. In *2010 IEEE International Conference on Data Mining*, pp. 481–490.

Toroslu, I. H. et G. Üçoluk (2007). Incremental assignment problem. *Information Sciences 177*(6), 1523–1529.

Zheng, W., L. Zou, X. Lian, D. Wang, et D. Zhao (2013). Graph similarity search with edit distance constraint in large graph databases. In *22nd ACM international Conference on information & knowledge management*, pp. 1595–1600.

Summary

In this work, we propose a new approach to detect anomalous graphs in a stream of directed and labeled heterogeneous graphs. Our approach uses a new representation of graphs by vectors. This representation is flexible and allows to update the graph vectors as soon as a new edge arrives. In addition, it is applicable to any type of graph and optimizes memory space. Moreover, it allows the detection of anomalies in real-time.