Anonymisation et Compression de Graphes : un Etat de l'Art

Alexis Guyot

Université de Bourgogne
alexis_guyot@etu.u-bourgogne.fr

5 février 2021

Résumé

Depuis l'émergence d'Internet et encore plus depuis l'arrivée du Web 2.0, aussi appelé Web Social, deux problématiques aux enjeux opposés se sont développées. D'un premier côté est apparu le besoin et l'envie d'analyser tous ces graphes sociaux riches en informations, autant de par leur structure que de par leur contenu. Et d'un autre, une éthique d'utilisation et des réglementations sont venues réguler et limiter l'accès aux informations de ces sources, dans le but de garantir aux utilisateurs une meilleure sécurité quant à l'utilisation de parties de leur vie privée. Ainsi, de l'existence du fossé qui tendait à se creuser entre ces deux problématiques du Web Social, est né un troisième domaine d'étude : l'anonymisation de graphes, ou comment permettre la meilleure utilisation possible de graphes sociaux, tout en garantissant un degré élevé d'anonymat à leurs principaux acteurs, les utilisateurs des outils sources. Dans cette synthèse, différentes familles de méthodes d'anonymisation seront présentées, et il sera discuté du ou des potentiels rôles que pourraient jouer les méthodes de compression sans pertes pour anonymiser des graphes.

I. Introduction

Avec près de 51% de la population mondiale qui les utilise et une croissance moyenne en nombre d'utilisateurs de plus de 10% entre l'année 2019 et 2020 [1], les réseaux sociaux sont devenus une part très importante, voire incontournable, des outils du Web. En interagissant entre eux de différentes façons, à travers des demandes d'amis, des partages, des réactions, ou autre, les utilisateurs créent et alimentent des connexions, qui peuvent ensuite donner naissance à des graphes qualifiés de sociaux. Pour rappel, les graphes sont des structures de données où un ensemble de sommets sont reliés entre eux par un ensemble d'arêtes. Leur notation courante est $G = (V, E)$, où G est un graphe constitué d'un ensemble V de sommets et un ensemble E d'arêtes. Dans le contexte d'un graphe social, on peut interpréter cette notation comme la définition d'une structure G qui représente $|E|$ interactions entre $|V|$ utilisateurs d'un environnement social. Une arête est généralement représentée par une paire de sommets (i, j) qui indique qu'il existe une connexion entre les sommets i et j, qui appartiennent tous les deux à l'ensemble V. Quand le graphe est non-orienté, cette connexion est considérée comme mutuelle, ce qui signifie que la présence de l'arête (i, j) dans l'ensemble E sous-entend l'existence de l'arête (j, i). Quand le graphe est orienté, cette hypothèse n'est pas vérifiée, et l'existence d'une arête entre deux sommets, qu'on appelle alors plutôt un arc, n'implique pas forcément l'existence de son inverse. La force d'une interaction entre deux utilisateurs peut être mesurée en attribuant un poids aux arêtes d'un graphe.

1. https ://www.forbes.fr/business/les-chiffres-fous-des-reseaux-sociaux/

On parle alors de graphes pondérés. Quand ce poids ne correspond pas à une valeur numérique mais à un mot, une lettre, un tableau, etc., on parle de graphes étiquetés. Les sommets comme les arêtes d'un graphe peuvent ou non contenir des attributs. Dans le contexte des réseaux sociaux, les sommets, qui représentent donc des utilisateurs, peuvent par exemple contenir un attribut d'âge, de genre, etc. Les arêtes, elles, peuvent par exemple avoir un attribut indiquant le type d'interaction qu'elles représentent. Les intérêts de telles structures sont extrêmement nombreux, et gravitent pour la plupart autour du fait que les graphes sociaux permettent de contenir et de représenter une population réaliste d'individus, ainsi que leurs rapports sociaux, dans des structures connues et pouvant être examinées au travers d'une machine. De telles études permettent ainsi et entre autres de mieux comprendre notre société, ce qui présente de très grands intérêts sociologiques, philosophiques ou encore commerciaux.

Cependant, une problématique majeure est à prendre en compte dans le cadre de toutes ces applications : les données, certes souvent accessibles de façon publique ou semi-publique dans le contexte d'utilisation de l'outil, restent avant tout des données privées. En effet, le Règlement Général sur la Protection des Données, ou RGPD, règlement européen entré en vigueur en 2018, définit une donnée privée comme « toute information se rapportant à une personne identifiée ou identifiable ». Avec l'utilisation de telles données vient ainsi des questions d'éthique : comment s'assurer que les utilisateurs, acteurs principaux de ces graphes sociaux, soient au courant de leur participation à ces corpus d'étude géants, et soient protégés en garantissant leur anonymat au sein des structures de données ? Si les questions du consentement sont normalement couvertes par les conditions d'utilisation des outils, les questions d'éthiques sont, elles, au cœur des préoccupations à la fois des producteurs et des consommateurs, afin de permettre la publication des données en toute sécurité. C'est la philosophie du "Pivacy by Design", principe clé du RGPD qui demande à toutes les entreprises récupérant et traitant des données de leurs utilisateurs d'inclure et de mettre en place des solutions de protection de la vie privée dès la conception des outils. Ainsi, et depuis plusieurs années, se développent des questionnements puis des méthodes autour de la question suivante : comment permettre le partage de graphes sociaux en garantissant le respect de la vie privée des utilisateurs et des réglementations, tout en préservant au maximum leur utilité dans le cadre d'analyses diverses et variées ? Une solution technique mise en avant pour arriver à ces fins est l'anonymisation de graphes.

L'anonymisation de graphes est un domaine d'étude dont l'objectif principal consiste à minimiser les risques de fuites de données privées sur les utilisateurs représentés dans des graphes sociaux, tout en maximisant l'utilité des structures selon plusieurs critères et caractéristiques essentiels à certaines analyses. Ces fuites peuvent prendre plusieurs formes : la découverte d'identité (*identity disclosure*), la découverte d'attributs (*content disclosure*) et la découverte de lien (*link disclosure*). On parle de découverte d'identité quand un sommet du graphe est correctement associé à une entité du monde réel. Par exemple, imaginons qu'un graphe soit extrait de Facebook pour détecter des communautés de membres au sein d'un club sportif. Un membre du club, identifié par la concaténation de son nom et de son prénom, représente un sommet du graphe, et le fait que deux personnes soient amis sur Facebook engendre la création d'un lien non-dirigé entre elles. Une technique d'anonymisation simple est mise en place par le réseau social pour préserver l'identité exacte de chaque personne. Pour cela, il a été décidé de remplacer chaque nom et prénom des utilisateurs par un nombre généré aléatoirement. Avec une telle méthode, il est toutefois toujours possible d'effectuer une découverte d'identité. Si une personne malveillante connaît le club en question et le nombre d'amis que possèdent certaines personnes réelles représentées dans le graphe, information qu'elle peut même potentiellement récupérer sur Facebook, alors celle-ci peut calculer le degré de chaque sommet et identifier le ou les candidats qui correspondent aux données qu'elle possède. Si elle réussit à associer ne serait-ce qu'un membre du club à un des sommets du

graphe, alors cette personne malveillante aura réussi à effectuer une découverte d'identité. Ce type de faille peut ensuite entraîner des découvertes en cascade. On peut réduire l'espace de recherche aux amis de la personne identifiée, puis effectuer la même manipulation pour découvrir d'autres identités, etc. L'inefficacité de cette méthode d'anonymisation naïve a été prouvée par Backstrom et al. [2]. On parle ensuite de découverte d'attributs quand des informations jugées sensibles, par exemple des données privées, sont associées à un utilisateur du réseau. Une découverte d'identité entraîne souvent une découverte d'attributs. Mais il est également possible que l'inverse soit vrai. Par exemple, le fait de découvrir qu'un sommet représente un homme de 21 ans qui habite à telle adresse peut amener à la découverte de son identité. Enfin, il est possible de vouloir empêcher la découverte de lien. Ce type de faille n'est pas cité dans l'état de l'art de la thèse de Nguyen [15], mais est notamment mentionné par Skarkala et al. [18] comme enjeu des méthodes d'anonymisation. On parle de découverte de lien quand l'existence d'une relation entre deux individus donnés peut être déterminée à l'aide d'une analyse du graphe. Ce type de faille peut notamment intervenir quand le cercle d'amis d'un utilisateur est considéré comme une donnée privée. La découverte de lien englobe aussi le fait de réussir à découvrir le poids d'une connexion entre deux utilisateurs.

Deux modes d'anonymisation peuvent être mis en place par les organismes qui publient des données sensibles. La première solution consiste à ne pas publier directement le graphe mais à proposer à la place un ou plusieurs points d'entrée qu'il est possible d'interroger à travers un certain nombre de requêtes prédéfinies. Le principe est ensuite d'effectuer les analyses sur le graphe non-anonymisé, puis de bruiter les résultats à renvoyer. Dans la littérature, cette méthode est qualifiée de mécanisme d'anonymisation interactif. Puisque notre problématique concerne plutôt la publication d'extraits de graphes en mode hors-ligne en tant que *datasets*, cette première solution ne sera pas plus évoquée dans la suite de cette synthèse. En effet, ces méthodes s'appliquent plutôt dans le contexte du deuxième mode d'anonymisation, le mécanisme non-interactif, qui consiste à publier un graphe dont les données ont été anonymisées avant la publication.

Bien que l'objectif principal des méthodes d'anonymisation soit de maximiser la sécurité des données d'un graphe, une contrainte indispensable que ceux-ci doivent également gérer est la préservation de l'utilité des données à des fins d'analyses. En effet, il est inutile de publier des données si aucune conclusion sur elles ne peut être proposée par la suite. Ainsi, la vraie difficulté de l'anonymisation réside dans le fait que ces deux forces, la sécurité des données et leur utilité, sont en réalité conflictuelles. Améliorer le respect de la vie privée des utilisateurs du graphe tend à diminuer l'utilité des données. Cette observation est souvent référée dans la littérature comme le *privacy/utility tradeoff*, ou en français le compromis vie privée/utilité. Cela implique qu'une méthode d'anonymisation n'est pas seulement évaluée ou comparée selon sa capacité à préserver la vie privée des utilisateurs du graphe, mais aussi sur le nombre de caractéristiques du graphe initial préservées après son application, ainsi que la qualité de cette préservation. Pour définir le niveau de sécurité d'un graphe anonymisé, les concepts de k-anonymat [13] sont très souvent utilisés. De manière générale, on considère qu'un graphe est k-anonyme s'il est impossible pour un individu malveillant de distinguer un de ses sommets avec $k-1$ autres. Cela implique que pour une caractéristique donnée sur le graphe, il n'existe pas moins de k correspondances. Par exemple, un graphe est dit 4-anonyme sur son degré si, pour chacune des valeurs de degré possibles, il existe au moins 4 sommets qui la possèdent. Plus formellement, imaginons qu'on souhaite toujours proposer une méthode qui empêche cet individu d'exploiter une des trois failles précédentes à l'aide des degrés d'un à la totalité des sommets du graphe $G = (V, E)$. L'individu malveillant va essayer d'attaquer le graphe en effectuant une requête structurelle Q, qui va retourner pour un sommet donné son degré. On considère que la méthode qui a permis la publication de G

propose une anonymisation de type k-degré si tous ses sommets sont k-candidats vis-à-vis de la requête Q. Pour compléter la définition, un sommet $v \in V$ est k-candidat à Q si au moins $k-1$ autres sommets de G, différents de v, retournent la même valeur que lui après application de Q. Une grande variété de k-anonymisations existent pour couvrir différentes possibilités de requêtes structurelles Q : k-voisinage, k-automorphisme, k-symétrie, k-isomorphisme, etc.

Les méthodes d'anonymisation de graphes peuvent être classifiées dans deux principales familles : l'anonymisation par généralisation et l'anonymisation par ajout de bruit. Dans le premier cas, les méthodes utilisent une notion de compression de graphes avec pertes de données, la généralisation, pour retirer astucieusement des éléments d'un graphe initial afin d'obtenir sa version anonymisée. Dans le second cas, qui fonctionne globalement à l'inverse du premier, les méthodes ajoutent certaines données au graphe d'origine, du bruit, jusqu'à ce que celui-ci soit assez modifié pour fournir un anonymat suffisant aux différents sommets. Dans la section II de cette synthèse, nous commencerons par étudier les deux grandes familles de méthodes d'anonymisation. Pour chaque, quelques méthodes seront détaillées pour illustrer différents points de vue sur la façon d'appliquer la généralisation et le bruitage. Ensuite, nous étudierons au travers de la section III une méthode de compression de graphes sans pertes de données, la méthode *WebGraph* [7], ainsi que deux de ses extensions. Après quelques explications sur les 3 techniques dans la sous-section i, nous discuterons dans la sous-section ii de quelques pistes qui pourraient être creusées afin de profiter de cette autre facette de la compression de graphes dans un contexte d'anonymisation. À travers cette synthèse, on cherche à identifier un potentiel rôle que pourraient jouer les méthodes de compression sans pertes pour achever de l'anonymisation de graphes.

II. Anonymiser un graphe

i. Anonymisation par généralisation

i.1 Introduction de la généralisation

Une première idée pour anonymiser les données d'un graphe consiste à utiliser le concept de généralisation. Celui-ci a à l'origine été développé dans le contexte de la compression avec pertes de graphes, et consiste grossièrement à produire une version résumée d'une structure d'origine. Quelques mots d'abord sur le domaine de la compression. Celui-ci regroupe un ensemble de méthodes aux objectifs divers et variés, qui partagent tout de même la même ligne directrice : réduire la taille qu'un graphe occupe en mémoire, à la fois pour lui permettre d'être chargé entièrement dans la mémoire vive d'une machine, mais aussi tout simplement pour pouvoir les stocker plus facilement sur certains appareils. Cela permet notamment de réduire le nombre d'opérations coûteuses en temps d'exécution, comme la lecture/écriture sur disque, ou les échanges de données entre machines distribuées. Un autre objectif principal de la compression de graphes est l'accélération des algorithmes et requêtes qui portent sur eux. En effet, un bon nombre d'algorithmes d'analyse et de traitement de graphes ne supportent pas très bien la montée en charge imposée par le traitement de plus grandes structures. À part cela, les techniques de compression peuvent également être utilisées pour répondre à beaucoup d'autres besoins, comme permettre la visualisation de grands graphes, réduire le bruit en leur sein, pour du clustering, de la classification, de la détection de communautés, etc. Tous ces objectifs, aussi nombreux que variés, montrent la grande utilité des méthodes de compression. Il n'est donc pas étonnant de les voir se spécialiser de nouveau pour couvrir un besoin comme l'anonymisation. La topologie parfois complexe des graphes à traiter est le premier défi à relever pour la compression. Les données en entrée représentent souvent un gros volume et peuvent aussi être très hétérogènes (texte, images,

...), bruitées, voire même incomplètes. Pour certain types de graphes, la majorité même lorsqu'on se réfère à des données réelles, les données peuvent également évoluer au cours du temps. Pour les méthodes qui se basent sur le retrait de données jugées inutiles au graphe, comme la généralisation par exemple, une autre problématique très importante concerne la définition de ce qu'est une donnée utile. Par exemple, préserver la structure des communautés dans la version compressée d'un graphe peut être une nécessité pour les utilisateurs qui souhaitent les étudier, mais peut également s'avérer complètement inutile pour un autre qui ne s'intéresse qu'à l'existence ou non d'un lien entre deux individus. La définition de l'utile et de l'inutile est une difficulté et un enjeu que partagent la compression et l'anonymisation de graphes. Pour finir, la dernière problématique à traiter lors de l'utilisation de méthodes de généralisation concerne les indicateurs à prendre en compte pour évaluer son utilité. En effet, la variété d'utilisations et d'objectifs possibles rend difficile la définition d'une valeur générique indiquant si la généralisation a été efficace ou non. Cette difficulté complique la comparaison du grand nombre de méthodes disponibles et rend moins aisé la création d'états de l'art comme proposé par [14].

De manière générale, deux principales familles d'algorithmes existent quand on parle de compression de graphes : les compressions avec et sans pertes. Parmi les méthodes de compression sans pertes, on retrouve par exemple celles basées sur *WebGraph* [7], qui seront abordées dans la section III.i. Parmi les méthodes de compression avec pertes, on retrouve la généralisation. Dans leur état de l'art, Liu et al. [14] recensent, expliquent et comparent 31 techniques de généralisation de graphes pour aboutir à une compression. L'étude est structurée en trois temps, d'abord avec une présentation des méthodes adaptées aux graphes simples, puis de celles adaptées aux graphes statiques étiquetés, et enfin de celles adaptées aux graphes dynamiques. Un graphe dynamique est une structure dont la topologie et/ou les données évoluent au cours du temps. L'étude des méthodes sur ce dernier type de graphe est ce qui constitue la contribution majeure de cet état de l'art par rapport aux autres déjà existants sur le domaine de la généralisation de graphes.

La généralisation de graphes est présentée par [14] comme un terme générique qui regroupe 4 familles principales d'algorithmes : la compression par regroupement de sommets ou d'arêtes, la compression des représentations internes des graphes, la compression par simplification et la compression par construction d'un modèle d'influence. Bien que toutes ces méthodes souhaitent atteindre le même but, il est difficile d'en considérer une meilleure que les autres à cause des nombreuses différences qu'elles présentent entre elles, vis-à-vis notamment du type de graphe qu'elles prennent en entrée, de leur caractéristiques algorithmiques, de la forme du résultat qu'elles retournent ou encore de l'objectif secondaire qu'elles souhaitent atteindre. Par exemple, certaines ne sont adaptées que sur des graphes dirigés ou non-dirigés, d'autres sur les deux. Certaines nécessitent la pré-définition de paramètres par l'utilisateur et/ou possèdent une complexité algorithmique linéaire, d'autres non, etc. Concernant les objectifs secondaires, on peut souhaiter compresser un graphe et conserver un certain nombre de propriétés pour des cas d'utilisation précis. C'est cette nuance qui différencie réellement la compression de la généralisation : la simple compression cherche à obtenir la plus petite représentation possible d'un graphe, alors que la généralisation cherche à obtenir la plus petite possible qui garantit la possibilité d'effectuer un certain nombre d'analyses déterminées à l'avance. Il peut s'agir par exemple du besoin de conserver certains patterns structuraux, certaines entités du réseau ou tout simplement la possibilité d'accéder rapidement à certaines mesures comme le diamètre ou la densité à travers des requêtes. En réalité, il est plutôt commun de considérer les deux concepts comme équivalents, puisque la généralisation implique toujours une compression des données, qui n'est juste pas forcément optimale. Dans un contexte d'anonymisation, la minimisation de la taille du graphe est remplacée par la maximisation d'un facteur de sécurité. On remarque cependant assez facilement le point d'accroche qui a entraîné l'association des deux idées : dans les deux cas, on cherche à

obtenir un compromis entre un besoin particulier et la bonne préservation de l'utilité du graphe.

Plutôt que d'effectuer une comparaison basée sur un indice de performance particulier, les auteurs de [14] font donc le choix de juste présenter les 31 méthodes et de les caractériser selon plusieurs critères, qui définissent le contexte dans lequel utiliser une technique plutôt qu'une autre. Ceux-ci sont les suivants :

— le type de graphe : simple, étiqueté ou dynamique ;
— les propriétés de topologie prises en charge : pondération et direction des arêtes, hétérogénéité ;
— les propriétés algorithmiques de la méthode : utilisation ou non de paramètres à fixer par l'utilisateur, complexité linéaire ou non ;
— le type de technique : par regroupement, par compression d'une représentation interne, par simplification ou par influence ;
— la nature de résultat retourné ;
— l'objectif secondaire à atteindre grâce à la compression : possibilité d'effectuer des requêtes efficacement, visualisation, détection d'entités et/ou de pattern, détection d'influence, etc.

L'étude de [14] est résumée à travers le tableau comparatif de la figure 1.

		Input Graph				Algorithmic Properties				
	Method	Weighted	Undirect.	Directed	Heterog.	Prm-free	Linear	Technique	Output	Objective
Static Plain Graphs	GraSS [LeFevre and Terzi 2010]	✘	✔	✘	✘	✘	✘	grouping	supergraph	query efficiency
	Weighted Compr. [Toivonen et al. 2011]	✔	✔	✔*	✘	✘	✘	grouping	supergraph	compression
	COARSENET[Purohit et al. 2014]	✔	✘	✔	✘	✘	✔*	grouping	supergraph	influence
	l_p-reconstr. Error [Riondato et al. 2014]	✔	✔	✘	✘	✘	✔	grouping	supergraph	query efficiency
	Motifs [Dunne and Shneiderman 2013]	✘	✔	✘	✔	✘	✘	grouping	supergraph	visualization
	CoSum [Zhu et al. 2016]	✔*[1]	✔	✘	✔	✔	✔	grouping	supergraph	entity resolution
	Dedensification [Maccioni and Abadi 2016]	✘	✔*	✔	✔	✔	✔*	(edge) grouping	sparsified graph	query efficiency
	VNM [Buehrer and Chellapilla 2008]	✘	✘	✔	✘	✘	✘	(edge) grouping	sparsified graph	patterns
	MDL Repres. [Navlakha et al. 2008]	✘	✔	✔*	✘	✔	✘	compression	supergraph	compression
	VoG[Koutra et al. 2014b]	✘	✔	✘	✘	✔	✔*	compression	structure list	patterns, visualiz.
	OntoVis [Shen et al. 2006]	✘	✔	✘	✔	✔	✔	simplification	sparsified graph	visualization
	Egocentric Abstr. [Li and Lin 2009]	✘	✘	✔	✔	✘	✘	simplification	sparsified graph	influence
	CSI [Mehmood et al. 2013]	✘	✘	✔	✘	✔	✘	influence	supergraph	influence
	SPINE [Mathioudakis et al. 2011]	✔	✘	✔	✘	✘	✘	influence	sparsified graph	influence
Static Labeled Graphs	S-Node [Raghavan and Garcia-Molina 2003]	✘	✘	✔	✘	✔	✘	grouping	supergraph	query efficiency
	SNAP/k-SNAP [Tian et al. 2008]	✘	✔	✔*	✘	✔	✔	grouping	supergraph	query efficiency
	CANAL [Zhang et al. 2010]	✘	✔	✔*	✘	✔	✘	grouping	supergraph	patterns
	Probabilistic [Hassanlou et al. 2013]	✔	✘	✔	✘	✔	✔	grouping	supergraph	compression
	Query-Pres. [Fan et al. 2012]	✘	✘	✔	✘	✘	✔	grouping	supergraph	query efficiency
	ZKP [Shoaran et al. 2013]	✘	✘	✔	✘	✔	✔	grouping	supergraph	privacy
	Randomized [Chen et al. 2009]	✘	✔	✘	✔	✔	✘	grouping	supergraph	patterns
	d-summaries [Song et al. 2016]	✘	✘	✔	✘	✘	✘	grouping	supergraph	query efficiency
	SUBDUE [Cook and Holder 1994]	✘	✔	✔	✔	✔	✘	compression	supergraph	patterns
	AGSUMMARY [Wu et al. 2014]	✘	✘	✔	✘	✔	✔	compression	supergraph	compression
	LSH-based [Khan et al. 2014]	✘	✘	✔	✘	✘	✔	compression	supergraph	compression
	VEGAS [Shi et al. 2015]	✔*	✘	✔	✘	✘	✔*	influence	supergraph	influence
Dynamic Graphs	NETCONDENSE [Adhikari et al. 2017]	✔	✔	✔	✘	✘	✘	grouping	temporal supergraph	influence
	TCM [Tang et al. 2016]	✔	✔	✔	✘	✘	✔	grouping	supergraph	query efficiency
	TimeCrunch [Shah et al. 2015]	✘	✔	✘	✔	✘	✔	compression	ranked list of temporal structures	temporal patterns, visualization
	OSNET [Qu et al. 2014]	✘	✔	✘	✘	✘	✔	influence	subgraphs of diffusion over time	influence
	Social Activity [Lin et al. 2008]	✘	✔	✘	✔	✘	✘	influence	temporal themes	influence, visualization

Y. Liu, T. Safavi, A. Dighe and D. Koutra

Figure 1 – *Comparaison des méthodes présentées dans l'étude [14]. Les * dans la colonne Linear indiquent une complexité sous-linéaire et dans les autres colonnes que la méthode n'est pas spécifiée pour cette propriété mais peut être étendue pour la prendre en charge.*

Les méthodes de généralisation basées sur de la compression de bits sont assez courantes et réputées dans le domaine. Le principe général de ces méthodes est de créer un résumé d'un graphe en minimisant le plus possible le nombre de bits nécessaires pour le représenter. Elles sont peu utilisées en anonymisation de par leur orientation trop axée sur la minimisation de la taille du graphe, qui n'est pas un objectif de ce domaine. Les méthodes de généralisation basées sur de la simplification (ou *sparsification*) consistent simplement à retirer les sommets et/ou arêtes

jugés les moins importants, afin de rendre un graphe de départ plus épars. Parmi les méthodes d'anonymisation qui utilisent ce principe, on peut rapidement citer *random sparsification* de Bonchi et al. [8], qui consiste à retirer aléatoirement des arêtes selon une certain probabilité. Ce type de généralisation est toutefois assez peu utilisé, car la distorsion qu'elles appliquent sur les graphes à anonymiser est très forte. Les méthodes de généralisation basées sur l'influence ont pour objectif de découvrir une description de haut niveau de la propagation d'influence au sein de grands graphes. Le problème de généralisation est alors transformé en un processus d'optimisation de la taille du graphe, dans lequel on essaye de maintenir au maximum les mesures d'influence calculées sur le graphe d'origine. Les méthodes de généralisation par regroupement (ou par agglomération), sont les plus populaires des quatre familles de techniques présentées par [14] et sont celles majoritairement utilisées par l'anonymisation. Leur principe général consiste à créer un nouveau graphe où chaque sommet représente un groupe de sommets du graphe de base et/ou chaque arête représente un groupe d'arêtes. Les agglomérations sont formées à partir d'une fonction de similarité à optimiser ou à partir de méthodes de clustering selon les cas. Dans la suite de cette synthèse, par souci de simplicité, nous nous référerons toujours à des méthodes par regroupement quand nous parlerons de généralisation.

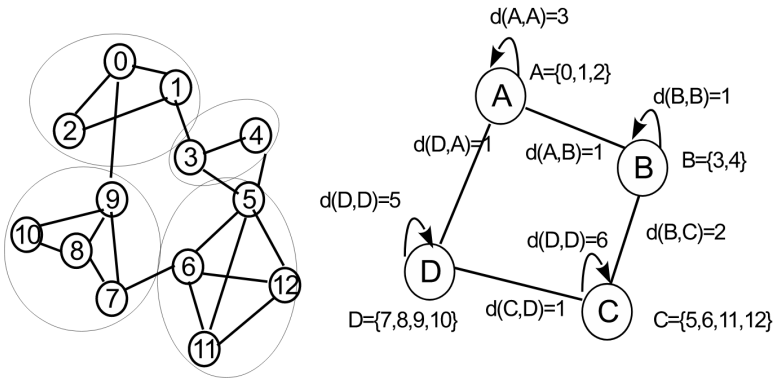


Figure 2 – *Généralisation d'un graphe par regroupement (exemple tiré de [15]).*

L'idée de base derrière la généralisation, comme présentée dans la figure 2, consiste à transformer un graphe de départ $G = (V, E)$ en un graphe résumé $G' = (V', E')$. L'ensemble V' contient des super-nœuds tous créés à partir d'un regroupement de nœuds appartenant à V. L'ensemble E' est lui constitué de super-arêtes créées et pondérées pour représenter les liens disparus lors du regroupement des nœuds en super-nœuds. Dans la figure 2, on remarque que le super-nœud A contient les sommets 0, 1 et 2. Trois arêtes sortent de A : (A,A), (A,B) et (A,D). On remarque que d(A,A) vaut 3, ce qui correspond aux trois arêtes (0,1), (0,2) et (1,2) qui ont disparu du graphe généralisé et qui séparaient toutes des sommets qui se retrouvent ensuite dans le super-nœud A. On note que d(A,B) vaut 1 car, dans le graphe d'origine, seule une arête séparait des sommets qui sont maintenant dans A ou dans B : le lien (1,3). Et ainsi de suite. On remarque que l'utilisation de base de la généralisation permet nativement de contribuer à la protection contre deux des trois principales failles de sécurité. Le concept de super-nœuds rend les sommets qu'ils contiennent indissociables les uns des autres, proposant ainsi une forme de k-anonymat et donc de protection contre la découverte d'identité. Les super-arêtes protègent quant à elles de la découverte de lien. En effet, les poids qu'elles possèdent traduisent la présence d'arêtes mais ne permettent pas de déterminer exactement à quels sommets du graphe d'origine celles-ci étaient attachées. Cette pondération représente en réalité la probabilité d'existence d'une arête reliant un sommet

appartenant au premier super-nœud à un sommet appartenant au second. Pour la super-arête (A,B), dont la pondération vaut 1, on sait juste qu'il existe dans le graphe d'origine une arête qui relie ou bien 0 et 3, ou bien 0 et 4, ou bien 1 et 3, etc. Il y a n arêtes possibles, avec n égal au produit des tailles des deux super-nœuds, et donc une probabilité de $1/n$ que deux sommets choisis au hasard dans les deux super-nœuds possèdent bel et bien un lien entre eux. Si un individu malveillant essaye de découvrir l'existence d'un lien entre un utilisateur se trouvant dans le super-nœud A et un autre dans le super-nœud B, qui regroupent respectivement 3 et 2 sommets, alors il ne pourra seulement déterminer qu'il n'y a qu'une chance sur 6 que cette arête existe réellement dans le graphe d'origine.

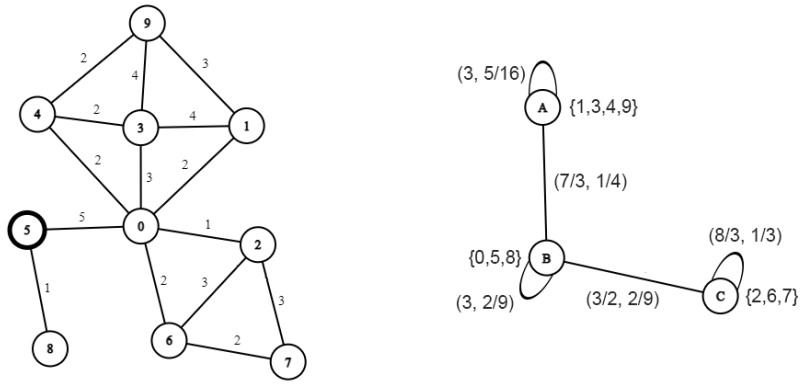
La plupart des méthodes de généralisation pour anonymiser un graphe sont en réalité des extensions de ce principe de base. Elles varient principalement entre elles en fonction de la technique utilisée pour créer les super-nœuds et la pondération à appliquer aux super-arêtes. Par exemple, Hay et al. [13] proposent juste d'utiliser ce principe avec un algorithme de recuit simulé pour trouver une approximation du regroupement optimal pour les super-nœuds. La valeur qu'ils essayent de maximiser est une estimation de la similarité maximale entre deux sommets. Dans la suite de cette section, nous détaillerons trois autres extensions de ce principe de base. La première méthode, présentée par Skarkala et al. [18], propose plutôt de se concentrer sur un haut niveau de sécurité, au détriment de l'utilité. À l'inverse, la deuxième méthode, présentée par Casas et Rousseau [10], propose de se concentrer sur la préservation de l'utilité. Ces deux méthodes, aux objectifs bien que très différents, possèdent un point commun négatif, qu'elles partagent aussi avec le principe de base, à savoir la non-prise en charge de la faille de sécurité que constitue la découverte d'attributs. Pourtant, il n'est pas rare du tout d'anonymiser des graphes qui en possèdent, puisque ceux-ci sont issus de réseaux sociaux, où les utilisateurs décrivent de façon assez détaillée leur profil. Campan et Truta [9] proposent alors la troisième méthode que nous traiterons, qui pour une fois prend en considération ces fameux attributs de sommets.

i.2 La sécurité au profit de l'utilité : l'approche de Skarkala et al.

Skarkala et al. proposent une méthode d'anonymisation [18] basée sur une généralisation, avec comme objectif de protéger les utilisateurs contre une découverte d'identité et de lien. Pour cela, ils veillent à proposer une anonymité de degré, de pondération et de voisinage dans certains cas. La méthode est spécifiée sur des graphes pondérés et non-orientés. Le principe est similaire à celui exprimé dans la figure 2, mais varie sur sa façon de pondérer les super-arêtes. En effet, le poids d'une super-arête est cette fois-ci défini comme la moyenne des arêtes du graphe initial qu'elle contient. Les auteurs recommandent également d'appliquer une légère variation aléatoire sur le poids obtenu, afin de garantir l'anonymité de pondération. Celle-ci vise à empêcher l'individu malveillant d'effectuer une découverte d'identité en ayant connaissance de la pondération du graphe initial. Cette définition s'oppose à celles des autres méthodes par généralisation de cette synthèse, qui définissent plutôt le poids comme le nombre d'arêtes que la super-arête regroupe.

À l'étape d'initialisation de la méthode, chaque sommet du graphe se voit attribuer un identifiant aléatoire et constitue un super-nœud. Ceux-ci sont ensuite successivement fusionnés entre-eux. Selon la taille du graphe et le temps d'exécution souhaité, [18] préconise différentes méthodes pour déterminer les super-nœuds à fusionner entre eux. La méthode la plus rapide, mais qui préserve le moins d'utilité, consiste à les choisir aléatoirement. Les deux autres méthodes consistent à choisir la fusion qui ou bien minimise la perte d'information, ou bien la maintient en dessous d'un certain seuil fixé à l'avance. La perte d'information en question est évaluée en observant la différence entre la pondération des arêtes dans le graphe initial et celle des super-arêtes correspondantes dans le graphe anonymisé. Pour être plus précis, celle-ci correspond à la

somme des différences au carré des poids que possèdent les arêtes dans le graphe initial et de celui que possède la super-arête qui les représente dans le graphe anonymisé. Les super-nœuds sont fusionnés entre eux jusqu'à ce que tous regroupent plus de k nœuds, afin de garantir cette fois-ci la k-anonymat de la méthode. Chaque super-nœud possède la liste des identifiants des sommets du graphe d'origine qu'il regroupe. Un deuxième attribut de pondération est ajouté à chaque super-arête, pour indiquer la probabilité qu'un lien existe vraiment entre deux sommets pris aléatoirement dans les super-nœuds de chaque extrémité. C'est de cette probabilité dont il était question quand il était dit sur l'exemple de la figure 2 qu'il y avait une chance sur 6 qu'une arête entre le super-nœud A et le super-nœud B existe vraiment. Pour insérer encore plus d'incertitude à la généralisation, la méthode prévoit une borne supérieure à cette probabilité, qui doit être fixée à l'avance. Celle-ci permet de cacher l'existence de zones localement denses en interactions au sein du graphe.



(a) *Graphe à anonymiser* **(b)** *Graphe anonymisé*

Figure 3 – *Anonymisation par la méthode de Skarkala [18]*

Par rapport à l'exemple de la figure 3, on voit que le super-nœud A représente les sommets 1, 3, 4 et 9, le super-nœud B les sommets 0, 5 et 8, et le super-nœud C les sommets 2, 6 et 7. Au minimum, chaque super-nœud regroupe 3 sommets, on peut alors considérer le graphe de droite comme 3-anonyme. On suppose que ce regroupement est celui qui minimise la perte d'information induite par l'agrégation, à savoir, pour rappel, la différence entre la pondération du graphe initial avec celle des super-arêtes correspondantes dans le graphe généralisé. Chaque super-arête possède deux attributs. Le premier est sa pondération, et vaut la moyenne des arêtes du graphe initial qu'elle représente. Si on prend l'exemple de la super-arête (A,B), son poids vaut 7/3. On remarque que celle-ci regroupe toutes les arêtes dont une des extrémités se trouve dans A et l'autre dans B. Cela correspond à trois liens : (4,0), (3,0) et (1,0). En calculant la somme de leur pondération, on obtient la valeur 7. Cela nous permet de conclure que la moyenne des arêtes représentées par (A,B), autrement dit sa pondération, vaut bien 7/3. Le deuxième attribut correspond à la probabilité qu'un lien existe réellement dans le graphe initial entre un sommet associé au super-nœud A et un autre associé au super-nœud B. Le premier super-nœud représente 4 sommets du graphe d'origine, le second 3. En théorie, avec une telle configuration, 12 arêtes avec une extrémité dans chaque super-nœud sont possibles. Or, en pratique, seules 3 d'entre elles existent réellement dans le graphe initial. La super-arête (A,B) ne représente que 3 des 12 arêtes théoriques possibles, sa probabilité associée est donc égale à 1/4.

Pour ce qui concerne la sécurité, on remarque que cette méthode permet d'obtenir quelques garanties. Avec la taille des super-nœuds et la probabilité des super-arêtes, un individu malveillant peut éventuellement déterminer le nombre de liens qui séparaient certains sommets, sans pouvoir savoir avec certitude des quels. Comme dit précédemment, on retrouve ici une forme de k-anonymat, qui protège le graphe anonymisé d'attaques basées sur les degrés et sur le voisinage de ses sommets. Puisque le poids des super-arêtes correspond à une moyenne, l'individu sera également incapable de récupérer avec certitude la valeur d'origine des pondérations. Dans ce sens, cette méthode fournit une bonne protection contre les failles de type découverte de liens. On ne peut pas vraiment lui reprocher l'absence de protection contre la découverte d'attributs, puisque la méthode ne considère volontairement pas ces éléments.

Dans leur article, les auteurs indiquent que leur méthode réussit à préserver de façon tout à fait correcte la possibilité de récupérer certaines caractéristiques du graphe initial : distribution des degrés, distribution des poids moyen des arêtes, distribution de la taille des chemins et volume du graphe. Il faut utiliser pour cela la fusion des super-nœuds qui minimise la perte d'information. En revanche, la méthode présente dans ce cas une très grande complexité algorithmique, ce qui la rend inutilisable sur de grands graphes.

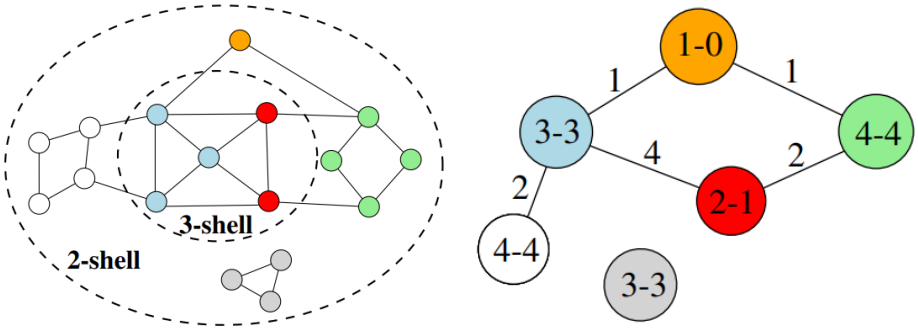
i.3 L'utilité au profit de la sécurité : l'approche de Casas et Rousseau

Contrairement à Skarkala et al. [18], Casas-Roma et Rousseau [10] proposent une méthode qui a pour but principal de préserver au maximum l'information structurelle des graphes, et notamment une caractéristique très importante pour leur analyse : la structure des communautés. La technique ne s'intéresse toujours pas aux cas des attributs et est spécifiée sur des graphes non-orientés et non-pondérés.

La méthode [10] considère et calcule deux notions lors de la phase de regroupement de sommets de la généralisation : la dégénérescence des graphes, via la définition de *k-shell*, et la similarité entre sommets. Le *k-shell* est une extension du principe de *k-core* pour un graphe. On considère qu'un sous-graphe G_{kc} est un *k-core* s'il contient l'ensemble des sommets d'un graphe d'origine G qui possèdent un degré supérieur à k, ainsi que toutes les arêtes qui les relient. Le *k-shell* G_{ks} correspond simplement au sous-graphe de G qui contient tous les sommets qui appartiennent à son k-*k-core* mais pas à son *(k+1)-core*, ainsi que toutes les arêtes qui les relient. Depuis le graphe G d'origine, il est possible d'attribuer à chaque sommet une valeur appelée le *core number* ou le *shell index*, et ce de manière efficace en temps linéaire grâce à la méthode [3]. Cet indice correspond à la plus grande valeur de k pour laquelle le sommet est présent dans le *k-shell* du graphe. Les auteurs remarquent que cette valeur permet d'évaluer à quel point le voisinage du sommet est fortement connecté, ainsi que l'engagement de l'utilisateur correspondant dans le réseau.

L'idée qu'ils développent est alors de d'abord regrouper entre eux les sommets qui partagent un même *shell index*, en partant du principe que l'agrégation aura peu d'impact sur la topologie générale du réseau. Ils justifient cette hypothèse par le fait que les sommets partagent une structure de voisinage et un rôle similaires. Ensuite, pour définir les groupes de nœuds à l'intérieur de chaque *k-shell*, la méthode recommande d'utiliser ou bien un algorithme de détection de communautés, ou bien la maximisation d'une mesure de similarité entre sommets. Dans l'article, ils étudient les performances obtenues avec les algorithmes *Multilevel* [4] et *Frastgreedy* [11], ainsi que la maximisation des mesures de similarités de *Manhattan* et *2-path*, qui sont présentées dans [19] comme les meilleures pour obtenir une valeur orientée sur le voisinage. Ces méthodes de regroupement sont donc lancées dans chaque *k-shell* pour obtenir un second découpage. Les sommets appartenant à une même communauté sont tous agrégés dans un même super-nœud,

sous condition que la taille de la communauté ne dépasse pas une certaine limite imposée pour éviter de se retrouver avec un trop gros super-nœud. Deux attributs sont ajoutés à chaque super-nœud pour indiquer la taille de la communauté correspondante et le nombre d'arêtes qui liaient deux sommets de la communauté. Comme dans la méthode de la figure 2 (principe général), les super-arêtes sont pondérées en comptant le nombre d'arêtes du graphe initial qui séparaient deux sommets se trouvant chacun dans une des deux extrémités.



(a) *Graphe à anonymiser* **(b)** *Graphe anonymisé*

Figure 4 – *Anonymisation par la méthode de Casas et Rousseau (exemple tiré de l'article [10]).*

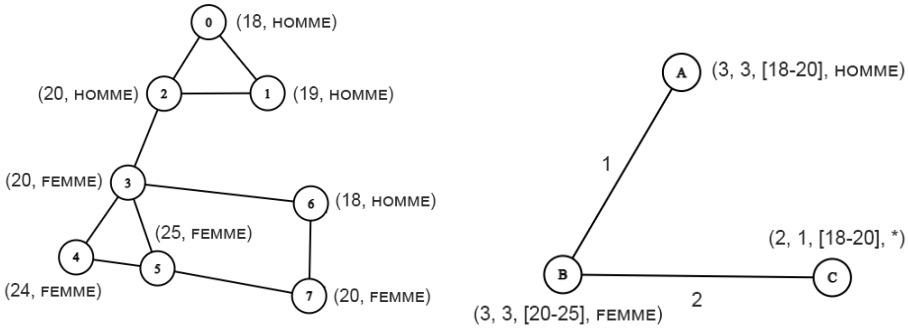
L'exemple de la figure 4 permet de mieux illustrer le processus d'anonymisation de la méthode. On remarque, sur le graphe initial à gauche, que le calcul des *shell indexes* a permis d'obtenir deux valeurs différentes, 2 et 3, qui sont respectivement attribuées à tous les sommets contenus dans chaque enveloppe. Pour les sommets verts, orange, blancs et gris, qui composent la couche *2-shell*, la détection de communautés les a séparés en 4 groupes, chacun représenté par une couleur différente. De même pour les sommets bleus et rouges de la couche *3-shell*. Le graphe de gauche, découpé en couches puis coloré, constitue le résultat obtenu à la fin des deux premières phases de l'algorithme [10]. Le graphe de droite est son résultat final après application de la troisième phase, qui consiste à construire les super-nœuds et les super-arêtes tout en leur ajoutant leurs attributs. Si on s'intéresse au super-nœud orange, on voit que ses attributs valent 1 et 0. Cela signifie que la communauté qui lui a donné naissance contenait 1 sommet et 0 arête interne. De la même manière, on peut déduire que le sommet gris correspond à une communauté de 3 individus reliés entre eux par 3 arêtes. La super-arête entre les super-nœuds rouge et vert indique que 2 arêtes étaient présentes dans le graphe d'origine entre des individus de la communauté verte et de la communauté rouge.

Intuitivement, cet exemple permet de voir à quel point la topologie du graphe initial semble être correctement préservée grâce à cette méthode. On remarque cependant aussi qu'elle ne permet pas forcément de garantir un k-anonymat, comme l'illustre le super-nœud orange qui est associé à seulement un seul sommet, qui pourra alors précisément être identifié par un individu malveillant si celui-ci possède les bonnes informations. Une évaluation des 4 méthodes de regroupement au sein des k-shell est proposée par [10] et révèle que peu importe le choix effectué, les résultats obtenus à la reconstruction permettent de correctement préserver un grand nombre de caractéristiques structurelles du graphe initial, comme la distribution des degrés de ses sommets, la distance moyenne entre deux nœuds, le diamètre, la moyenne harmonique des plus courts chemins, la transitivité et, comme souhaité, la structure des communautés. Cette dernière caractéristique est évaluée en étudiant les différences de classifications faites sur le graphe initial et sur le graphe anonymisé. On peut quand même noter une variation de la précision obtenue

entre les méthodes de détection de communautés et celles basées sur la similarité en fonction du type de caractéristique qu'on étudie. Pour la préservation des communautés, les méthodes de regroupement basées sur l'utilisation d'algorithmes de détection de communautés sont meilleures. Pour les autres propriétés structurelles, les méthodes basées sur la comparaison de similarité entre sommets sont meilleures, avec une précision légèrement plus grande avec la similarité de *Manhattan*.

Casas et Rousseau proposent avec cette méthode une manière de généraliser des graphes tout en préservant un haut niveau d'utilité. Le niveau de sécurité qu'ils proposent est cependant bien moins bon que celui offert par [18]. Il est notamment assez surprenant de noter qu'elle ne présente pas de forme de k-anonymat garantie dans tous les cas, qui est pourtant une base que s'assure de respecter la plupart des méthodes d'anonymisation basées généralisation. Cela se justifie par le fait de ne pas vouloir regrouper ensemble des groupes qui sont localement distants afin de ne pas trop altérer la topologie générale du graphe. Comme pour le principe de base expliqué dans la première sous-section, la méthode permet tout de même de se protéger un minimum contre la découverte de liens, puisqu'il est impossible de dire entre quels sommets des super-nœuds se trouvaient les arêtes du graphe initial. Tant que le premier attribut de tous les super-nœuds possède une valeur supérieure ou égale à 2, on a tout de même affaire à une propriété de k-anonymat, qui fournit toujours les mêmes protections contre la découverte d'identité.

i.4 Anonymiser les graphes avec attributs : l'approche de Campan et Truta



(a) *Graphe à anonymiser* **(b)** *Graphe anonymisé*

Figure 5 – *Anonymisation par la méthode SNGA [9]*

Campan et Truta présentent une méthode visant à fournir une anonymisation k-voisinage appelée *SaNGreeA (SNGA)* [9] (figure 5). La méthode est spécifiée pour des graphes non-orientés et enfin avec attributs. Elle se compose à la fois d'une technique d'anonymisation des attributs, via une méthode de généralisation de leurs valeurs, et des informations structurelles, via une technique de clusterisation pour déterminer des groupes disjoints de sommets pour les super-nœuds.

Chaque super-nœud possède donc des attributs qui représentent à la fois les informations propres au cluster (sa taille et son nombre d'arêtes intra-cluster), et les valeurs généralisées de chaque attribut des sommets du graphe initial. La méthode [9] distingue deux cas pour résumer les valeurs des attributs des nœuds. Si l'attribut en question est de type catégorie, alors sa généralisation se base sur une connaissance du domaine et consiste à choisir la première sur-catégorie capable d'englober toutes les valeurs. Celle-ci peut être définie ou non. Par exemple, les valeurs {France, Angleterre, Allemagne} peuvent être résumées par la sur-catégorie définie

Europe. En revanche, il n'existe parfois pas de terme pour qualifier l'ancêtre commun de certaines valeurs. Par exemple, des informations de genre {homme, femme}. Dans ce cas, on peut représenter la sur-catégorie par un symbole comme *. Si l'attribut est de type numérique, alors sa valeur généralisée correspond à l'intervalle entre la plus petite valeur du cluster et la plus grande. Par exemple, pour un attribut contenant l'âge des utilisateurs et prenant pour un cluster donné les valeurs {18, 18, 20, 25}, alors sa valeur généralisée est [18,25]. Par rapport au graphe de droite de la figure 5, on peut alors apprendre que le super-nœud A regroupe 3 sommets du graphe original, que ceux-ci étaient reliés entre eux par 3 arêtes et que l'un d'entre eux était lié à un des sommets regroupés dans le super-nœud B. Les trois sommets de A possédaient deux attributs, dont le premier contenait des valeurs numériques comprises entre 18 et 25 et le second des valeurs catégoriques égales à "Homme" pour tous les trois. Si on s'intéresse au super-nœud C, on voit que tous les sommets qu'il regroupe ne possédaient, eux, pas tous la même valeur pour le second attribut. Ceci est matérialisé par le symbole *.

Pour créer leurs clusters, Campan et Truta introduisent deux mesures de perte d'information. La première évalue la quantité d'information perdue lorsque les valeurs des attributs de deux nœuds sont généralisées. La seconde évalue la quantité d'information structurelle perdue si les deux sommets ne sont pas ajoutés dans le même cluster. Deux nœuds sont ajoutés au même cluster par un algorithme glouton si les deux quantités d'information perdue ne dépassent pas un certain seuil jugé raisonnable. Avant de terminer la phase de clusterisation, l'algorithme s'assure que chaque groupe contient au moins k sommets, afin de respecter la condition de k-anonymat. Les clusters sont ensuite transformés en super-nœuds et les valeurs de leurs différents attributs généralisées. Les super-arêtes sont finalement ajoutées entre les super-nœuds et les pondérations sont déterminées comme dans la méthode de base présentée dans la première sous-section.

Parmi les inconvénients de cette méthode, on peut citer le fait qu'elle soit spécifiée pour des graphes non-orientés et où tous les liens possèdent la même sémantique. Cette restriction sur la topologie des graphes l'empêche d'être utilisable dans certaines situations, par exemple sur un jeu de données représentant les interactions entre utilisateurs sur le réseau social Twitter. On peut également critiquer le fait qu'aucune méthode ne soit préconisée pour les attributs dont les valeurs ne sont ni numériques ni catégoriques (prénoms, noms, etc.) Enfin, l'article ne fait pas mention d'ordre de complexité. Puisque la création des clusters passe par une pré-évaluation de la quantité d'informations perdue par chaque aggrégation, il est possible que la complexité de la méthode soit assez élevée et puisse poser problème sur de grands graphes. Elle présente toutefois quand même quelques avantages. D'abord, par rapport à [13], elle prend en charge et se base sur les attributs des sommets, qui sont évidemment très importants dans un contexte d'anonymisation de graphes. On peut même considérer que cette méthode permet de prendre en charge les trois principales failles de sécurité au cœur de l'anonymisation. En effet, on retrouve une fois plus le respect du k-anonymat, qui offre une bonne protection contre la découverte d'identité et de lien, mais aussi la généralisation des valeurs, qui empêche d'associer à chaque utilisateur la valeur précise de son attribut potentiellement sensible. En plus de cela, elle vise à minimiser à chaque agrégation la perte d'informations structurelles et de contenu, ce qui garantit une assez bonne préservation de l'utilité du graphe de départ. Par association de ces deux avantages, on peut considérer que la méthode de Campan et Truta est à la fois pertinente et complète pour achever de l'anonymisation.

i.5 Conclusion sur les méthodes d'anonymisation basées généralisation

Pour conclure, les méthodes [13], [9], [18] et [10] nous ont permis de découvrir les idées clés derrière l'utilisation de la généralisation de graphes pour anonymiser. Dans certains cas, les auteurs essayent de maximiser la sécurité apportée par l'anonymisation, dans d'autres l'utilité

du résultat à publier. Par rapport à la synthèse, on peut regretter l'absence d'idées pour traiter certaines propriétés très importantes lors de l'étude de graphes issus de réseaux sociaux, comme l'orientation des liens par exemple. Seule la méthode [9] prend en charge les attributs des sommets, qui est aussi un point très important avec ce type de graphes. On peut toutefois remarquer que cette technique complète d'anonymisation sépare bien la phase de généralisation des attributs de celle de la structure du graphe, ce qui pourrait permettre sa réutilisation pour étendre les autres méthodes. En effet, puisque celles-ci ne sont pas incompatibles entre elles, il est tout à fait possible d'imaginer une fusion qui profiterait des avantages de sécurité apportés par [18], notamment au niveau de la pondération des super-arêtes, mais aussi de ceux concernant la préservation de la structure du graphe apportés par [10], avec l'utilisation des *k-shell* et des mesures de similarité ou des algorithmes de communautés, tout en proposant finalement une généralisation d'attributs inspirée de [9] et étendue pour prendre en charge d'autres types de valeurs. En poussant encore plus loin, il est même possible de récupérer l'idée d'utiliser un algorithme de recuit simulé, comme proposé par [13], pour maximiser l'utilité du résultat. Les méthodes présentées dans cette sous-partie possèdent une complexité algorithmique assez importante, notamment [18] et [9]. Cependant, ceci est à nuancer avec le fait qu'on cherche à découvrir des techniques visant à publier des extraits de graphes issus de données réelles. Cela implique donc que la taille de ceux-ci peut être adaptée et que leur anonymisation peut être faite de façon hors-ligne, sans exigence de temps d'exécution réel ou quasi réel.

Pour conclure sur l'idée d'utiliser la généralisation de graphes pour anonymiser, on peut noter que celle-ci propose des avantages conséquents, notamment le fait qu'elle permet nativement de couvrir deux des trois principales failles de sécurité combattues par l'anonymisation : la découverte d'identité, si légèrement adaptée pour ne pas permettre des regroupements d'un seul sommet, et la découverte de lien. L'utilisation de la généralisation permet de quand même profiter d'une réduction conséquente de la taille du graphe, ce qui peut être un atout de taille pour faciliter les analyses et traitements qui auront ensuite lieu sur les graphes anonymisés. Un inconvénient majeur réside toutefois dans le fait que la structure du graphe est profondément modifiée. Même si certaines caractéristiques d'origine peuvent être déduites à partir des différents attributs parsemés à travers le résultat, il est difficile, si ce n'est impossible, d'utiliser exactement les mêmes méthodes d'analyse et de traitement sur des graphes anonymisés par généralisation que si on avait dû faire les mêmes manipulations sur le graphe initial. Cela peut être un obstacle dans le cadre de certaines études. L'application de ce type d'anonymisation rend même carrément impossibles certaines analyses comme la recherche de cliques. Il est donc nécessaire de bien cerner les besoins des analystes qui se serviront des graphes anonymisés avant d'utiliser cette famille de méthodes d'anonymisation.

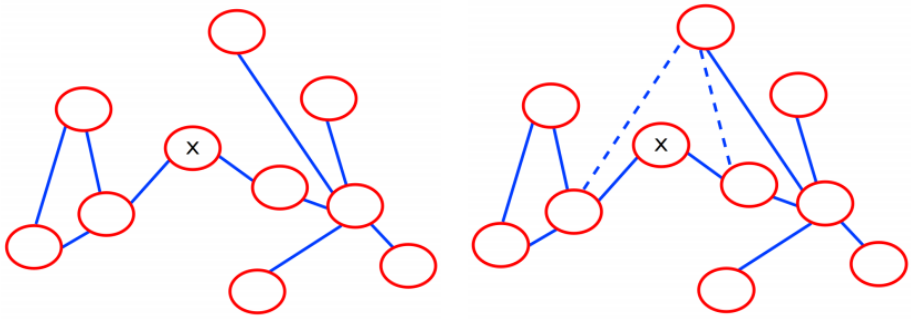
ii. Anonymisation par ajout de bruit

Bien que les méthodes d'anonymisation par généralisation représentent une bonne partie de la littérature concernée par ce sujet, d'autres techniques basées sur d'autres idées que la compression de graphes existent également. Au contraire des précédentes, celles-ci consistent plutôt à ajouter des "fausses données" (du bruit) jusqu'à atteindre un niveau de sécurité particulier. Les deux principales familles d'anonymisation s'opposent donc sur ce point-là. Deux écoles existent ensuite pour créer ce bruit. Dans un premier cas, on modifie le graphe d'origine de façon déterministe, en appliquant un certain nombre de règles qui, d'une exécution à l'autre, produiront toujours le même résultat. Dans le second cas, on injecte et/ou utilise des probabilités au sein du graphe pour produire un ensemble de résultats possibles. Dans cette seconde sous-partie consacrée à l'anonymisation, nous détaillerons quelques méthodes pour chaque type d'anonymisation par

ajout de bruit.

ii.1 Bruiter de manière déterministe : l'approche de Feder et al.

Une première façon d'ajouter du bruit au sein d'un graphe consiste à appliquer un certain nombre de règles déterministes. Le principe général est alors de modifier suffisamment le graphe de départ pour garantir un certain nombre de propriétés de sécurité, tout en minimisant la perte d'information.



(a) *Graphe à anonymiser* **(b)** *Graphe anonymisé*

Figure 6 – *Anonymiser en ajoutant des liens.*

Feder et al. [12] présentent une méthode qui consiste à produire un graphe anonymisé résistant aux attaques de k-voisinage. Cela implique qu'un individu mal intentionné possédant n'importe quelle information de voisinage sur un ou plusieurs sommets du graphe soit totalement incapable d'identifier moins de k correspondances. Le principe général de leur technique consiste à ajouter un certain nombre d'arêtes au graphe, le minimum possible, de sorte à préserver un maximum d'utilité tout en vérifiant la condition de k-voisinage. Celle-ci se différencie des techniques vues précédemment par le fait qu'aucune arête n'est supprimée en échange de celles ajoutées. La méthode est spécifiée pour des graphes non-orientés, non-pondérés et sans attributs. Feder et al. définissent une extension au principe de k-anonymat, la (k,l)-anonymité. Un graphe $G = (V, E)$ est considéré comme (k,l)-anonyme si pour chaque sommet $v \in V$, il existe au moins k autres sommets différents de v qui possèdent l voisins en commun avec lui. Les auteurs présentent alors deux problèmes liés à cette nouvelle définition : la (k,l)-transformation faible et la (k,l)-transformation forte.

On peut résumer la (k,l)-transformation faible comme le fait de trouver le nombre minimal d'arêtes à ajouter dans un graphe pour atteindre la (k,l)-anonymité, et la (k,l)-transformation forte comme le fait de trouver les meilleures arêtes à ajouter afin d'atteindre cette dernière avec le minimum de perturbation possible. Le second problème est alors juste une restriction du premier pour garantir encore plus de sécurité. Pour effectuer une (k,l)-transformation forte, le nombre de voisins en commun que doit avoir le sommet v avec les k autres sommets du graphe ne doit plus prendre en compte les liens qui ont été ajoutés par anonymisation. Cela implique, dans ce cas, de réduire la liste des voisins de v avant de vérifier la condition, de sorte à ce qu'elle soit identique à celle qu'il possède dans le graphe initial. Dans le graphe de gauche de la figure 6, si un individu malveillant connaît l'identité d'un des utilisateurs réels du graphe et sait qu'il est relié à un triangle, alors il pourra très facilement comprendre que cet utilisateur correspond au sommet x. Une transformation faible possible en (4,1)-anonymité pour résoudre ce problème est représentée dans le graphe de droite. En ajoutant la première arrête à gauche, on relie le sommet du haut au

triangle, ce qui le rend candidat possible à la description que possède l'individu malveillant. En ajoutant la seconde, celle de droite, on crée un nouveau triangle dans le graphe, dont le sommet en bas à droite est lui même relié à trois autres nœuds, qui deviennent ainsi eux aussi candidats potentiels. Par rapport à la requête structurelle Q, qui retourne la liste des sommets directement reliés à un triangle, le graphe de droite contient 4 correspondances. On peut alors dire qu'il est (4,1)-anonyme.

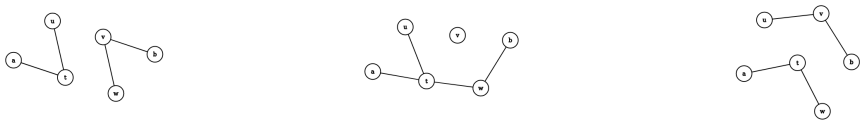
Dans leur article, Feder et al. observent que pour certaines valeurs de k et de l, il est possible de résoudre les deux types de transformation en temps polynomial en ajoutant les arêtes qui permettent d'obtenir le plus rapidement possible certaines caractéristiques de topologie. Par exemple, ils remarquent qu'un graphe est forcément (2,1)-anonyme si chacun des sommets est soit membre d'un triangle, soit voisin avec un sommet de degré supérieur à 3, soit au milieu d'un chemin de taille 5. Ils proposent alors un algorithme qui permet d'obtenir la (2,1)-transformation faible ou forte d'un graphe. Celui-ci utilise deux parcours des sommets. Dans un premier temps, une valeur de déficit est attribuée au nœud. Celle-ci est définie selon une liste de critères qui étudient la structure de son voisinage. Dans un second temps, les arêtes sont ajoutées selon un schéma particulier qui associe les sommets en fonction de leur déficit et qui change si la transformation recherchée est faible ou forte. Les principes vus sur les cas spéciaux sont ensuite généralisés, d'abord pour la (k,1)-anonymité puis la (k,l)-anonymité, toujours faible ou forte. Ces problèmes sont résolus à travers des algorithmes gloutons qui cherchent successivement une arête ou un triplet d'arêtes qui réduisent le plus ce qu'ils appellent l'anonymité résiduelle. Il s'agit en réalité du nombre de voisins en commun qui manquent aux sommets du graphe qui ne respectent pas encore la (k,l)-anonymité.

Le principal avantage de la méthode [12] est sa capacité à offrir une protection contre les découvertes d'identité et de lien très modulable selon les besoins grâce à ses deux paramètres. Celle-ci peut être très fine comme plus large, ce qui lui permet de s'adapter facilement en fonction des cas et du degré d'utilité à préserver. De plus, la méthode est applicable en temps polynomial dans un grand nombre de cas. En revanche, de nombreuses caractéristiques structurelles sont impactées par son application. On peut notamment citer la distribution des degrés, la longueur des plus courts chemins, la structure des communautés et d'autres. À part ces quelques informations, aucune donnée du graphe d'origine n'est perdue. Le graphe anonymisé possède exactement les mêmes individus que celui d'origine et toutes les connexions présentes dans ce dernier sont conservées. Cela signifie ainsi que certaines structures comme les cliques sont préservées, ce qui peut être d'un très grand intérêt dans le cadre de certaines analyses et ce qui démarque cette méthode de toutes les autres de cette synthèse.

ii.2 Bruiter de manière probabiliste : les approches de Boldi et al. et de Nguyen

Plusieurs méthodes existent pour anonymiser en créant du bruit de manière probabiliste. Une première possibilité, présentée à travers la figure 7, consiste à ajouter et à retirer aléatoirement des arêtes au graphe de départ. En ajoutant k faux liens et en retirant k vrais liens, le nombre total d'arêtes est préservé. Cependant, le degré des sommets est lui très probablement modifié. Certaines autres méthodes, comme *random perturbation* de Bonchi et al. [8], ne préservent même parfois aucune de ces deux caractéristiques. La technique consiste à retirer aléatoirement des arêtes selon une certaine probabilité et ensuite d'en ajouter de nouvelles selon une autre probabilité calculée à partir de la première. L'utilité des données par rapport au degré d'anonymisation est estimée grâce à la notion d'entropie développée par Shannon [2]. Pour préserver le degré des sommets et le nombre d'arêtes, il est possible de plutôt inverser un certain nombre de voisins

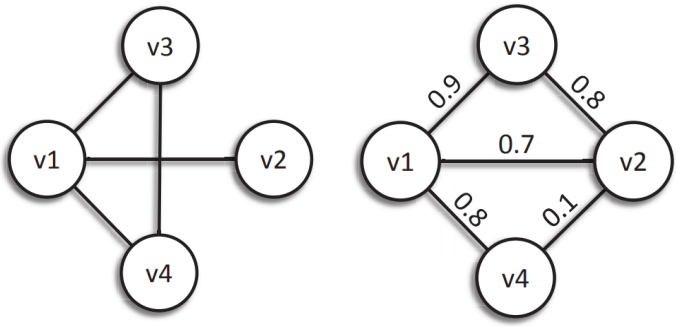
2. https ://fr.wikipedia.org/wiki/Entropie_de_Shannon



(a) *Graphe à anonymiser* **(b)** *Bruiter en ajoutant/supprimant des arêtes* **(c)** *Bruiter en échangeant des arêtes*

Figure 7 – *Anonymiser en créant du bruit de manière déterministe*

entre un nombre aléatoire de sommets. Si on considère les sommets t et son voisin u ainsi que v et son voisin w, tous choisis aléatoirement au sein du graphe à anonymiser, cette technique préconise de retirer les liens entre t et u et entre v et w puis d'en ajouter deux nouveaux entre t et w et entre v et u. De cette façon, le nombre total de liens est bien préservé, tout comme le degré de chacun des sommets. Comme l'indique Nguyen dans sa thèse [15], ces quelques méthodes simples présentent trois inconvénients majeurs : 1) elles ne garantissent pas la satisfaction d'un k-anonymat ; 2) la sécurité n'est accordée qu'à quelques sommets ; 3) elles ont un impact sur les valeurs propres de la matrice d'adjacence du graphe. Or, ces dernières sont intimement liées à plusieurs caractéristiques très importantes de la topologie d'un graphe, comme par exemple son diamètre, ses communautés, la longueur de ses plus courts chemins, etc. Certaines extensions ont donc été présentées pour éviter ce désagrément, comme les méthodes *Spctr Add/Del* et *Spctr Switch* de Ying et Wu [20]. Une fois les valeurs propres préservées, il devient possible d'effectuer une reconstruction spectrale depuis le graphe anonymisé. Celle-ci permet d'obtenir un graphe différent mais possédant des caractéristiques très proches de celles du graphe original.



(a) *Graphe à anonymiser* **(b)** *Graphe anonymisé*

Figure 8 – *Anonymiser en injectant de l'incertitude.*

Boldi et al. [5] proposent eux une méthode différente qui tente de préserver une meilleure utilité des données anonymisées. Le postulat de base consiste à voir l'ajout et la suppression d'arêtes comme une variation de leur probabilité d'existence de 0 à 1, et inversement. Les auteurs proposent alors de ne plus se contenter de faire varier cette probabilité de façon binaire, mais

de lui permettre de posséder n'importe quelle valeur réelle comprise dans cet intervalle. L'idée générale est alors d'impacter la certitude pour un individu malveillant concernant l'existence ou non de toutes les arêtes du graphe.

Pour cela, Boldi et al. [5] définissent le concept de graphe incertain $G' = (V, p)$, avec :

— V un ensemble de sommets, identique à celui du graphe d'origine $G = (V, E)$;
— p une fonction qui prend en entrée l'ensemble théorique de toutes les paires de sommets possibles de G, appelé V_2, et retourne en sortie une liste de probabilités comprises entre 0 et 1 pour chaque arête théorique du graphe. Sur un graphe non-orienté, la taille de V_2 vaut $n(n-1)/2$, avec $n = |V|$.

Pour réduire la complexité de la méthode sur de plus grands graphes, les auteurs proposent plutôt de ne travailler que sur un sous-ensemble de V_2, qu'ils appellent E_C, avec C une constante supérieure à 1 et égale à $|E_C|/|E|$ (ce qui implique que $|E_C| > |E|$). Ils considèrent ainsi juste que toutes les paires de sommets qui ne sont pas contenues dans E_C possèdent une probabilité nulle, ce qui permet de réduire la taille des données en entrée de leur algorithme. Afin d'évaluer le niveau de sécurité de leur méthode, Boldi et al. proposent le concept d'*obfuscation*. Celui-ci est défini à partir de tout un ensemble de sous-concepts :

— $W(G')$ correspond à l'ensemble des mondes possibles W du graphe incertain, soit l'ensemble des sous-graphes certains (V, E_W) qu'il est possible de construire avec $E_W \subseteq E_C$;
— $X_v(\omega)$ correspond à la probabilité qu'une propriété P du sommet v, par exemple son degré, possède la valeur ω dans l'ensemble des mondes possibles. $X_v(\omega)$ vaut alors la somme des probabilités de chaque configuration. Par exemple, sur la figure 8, on s'intéresse à la probabilité que le sommet $v2$ puisse posséder comme degré la valeur 1 dans l'ensemble des mondes possibles $W(G')$. Trois mondes possibles existent pour cette configuration. Dans le premier, seule l'arête $(v3, v2)$ a été conservée, dans le deuxième, seule l'arête $(v1, v2)$, et dans le dernier seule l'arête $(v4, v2)$. Pour la première, l'arête $(v3, v2)$ est conservée, ce qui correspond à une probabilité de 0.8, l'arête $(v3, v1)$ n'a pas été conservée, ce qui correspond à une probabilité de (1 - 0.7) ou 0.3, et l'arête $(v3, v4)$ non plus, probabilité de (1 - 0.1) ou 0.9. La probabilité totale de cette configuration correspond au produit de ces trois valeurs, soit $0.8 \times 0.3 \times 0.9 = 0.216$. On fait de même pour les deux autres configurations : $(1 - 0.8) \times 0.7 \times (1 - 0.1) = 0.126$ et $(1 - 0.8) \times (1 - 0.7) \times 0.1 = 0.006$. Finalement, on obtient que $X_{v2}(\omega) = 0.216 + 0.126 + 0.006 = 0.348$, avec ω correspondant à une valeur de degré égale à 1. Cela signifie que si on reconstruisait aléatoirement un graphe certain à partir de G', alors il y aurait 34.8% de chance que le sommet $v2$ possède un degré égal à 1 dans celui-ci ;
— $Y_\omega(v)$ permet d'évaluer la force de la protection apportée par la méthode. En effet, celle-ci représente les chances que possède un individu malveillant de correctement identifier un sommet du graphe en possédant la propriété structurelle bien précise ω. Elle s'obtient en divisant la probabilité $X_v(\omega)$ par la somme des valeurs de cette même probabilité pour chaque sommet du graphe. Vis-à-vis de la figure 8, imaginons que l'individu malveillant connaisse l'identité d'un des sommets du graphe (qui est en réalité $v1$, information qu'il essaye de déterminer) et sache que celui-ci possède un degré égal à 3. En utilisant la même méthode que précédemment, on obtient l'information que le sommet $v1$ possède un degré égal à 3 avec une probabilité de 0.504 dans l'ensemble des mondes possibles ($X_{v1}(\omega) = 0.504$). De même, on apprend que $X_{v2}(\omega) = 0.056$, $X_{v3}(\omega) = 0$ et $X_{v4}(\omega) = 0$. La valeur $Y_\omega(v1)$ permet d'évaluer la probabilité que le sommet $v1$ soit l'utilisateur dont l'individu malveillant possède l'identité. On obtient sa valeur en divisant $X_{v1}(\omega)$ par la somme des $X(\omega)$ pour chaque sommet du graphe. Ainsi, $Y_\omega(v1) = \frac{X_{v1}(\omega)}{X_{v1}(\omega)+X_{v2}(\omega)+X_{v3}(\omega)+X_{v4}(\omega)} = \frac{0.504}{0.504+0.056+0+0} =$

0.9. Grâce à cette mesure, on sait qu'en possédant l'information ω (qui correspond au degré égal à 3), l'individu malveillant réussira à correctement associer l'utilisateur qu'il connaît et le sommet $v1$ avec une probabilité de 90% ;

— on dit qu'un graphe incertain G' *k-obfuscate* un sommet v du graphe initial G par rapport à une propriété structurelle P si l'entropie de Shannon de la distribution des valeurs de Y_P pour chaque sommet du graphe incertain est supérieure ou égale à $log_2 k$. Cette valeur d'entropie est obtenue en utilisant la formule suivante : $-\sum_{i=1}^{n} Y_P(i) \times log_2 Y_P(i)$. Précédemment, nous avons calculé $Y_\omega(v1) = 0.9$. Les valeurs restantes pour ω sont $Y_\omega(v2) = 0.1$, $Y_\omega(v3) = 0$ et $Y_\omega(v4) = 0$. L'entropie de Y_P, avec P toujours le degré d'un sommet dont la valeur ω vaut 3, vaut alors $-0.9 \times log_2(0.9) - 0.1 \times log_2(0.1) \approx 0.469$. La valeur de k est alors comprise entre 0 et 1, ce qui indique que le graphe incertain G' n'offusque pas assez les sommets du graphe initial G qui possèdent un degré égal à 3, c'est-à-dire le sommet $v1$;

— finalement, un graphe incertain G' est considéré comme une $(k, \epsilon) - obfuscation$ d'un graphe initial G vis-à-vis d'une propriété donnée P s'il *k-obfuscate* au moins $n(1 - \epsilon)$ de ses sommets. Si on calcule l'entropie de chaque autre valeur ω possible pour la propriété P dans G, ce qui correspond aux valeurs de degré 1 (v2) et 2 (v3, v4), on obtient respectivement environ 1.688 et 1.742. Ces deux valeurs sont comprises entre $log_2(3)$ et $log_2(4)$, ce qui signifie que les sommets v2, v3 et v4 sont 3-offusqués par le graphe incertain G'. On remarque alors que 3 des 4 sommets du graphe initial sont concernés par cette définition, ce qui donne une valeur $\epsilon = -\frac{3}{n} + 1 = 0.25$. On peut alors dire que G' est une (3,0.25)-*obfuscation* de G.

Toutes ces définitions sont ensuite nécessaires pour comprendre l'algorithme [5] qui, étant donné un graphe initial G, un niveau d'*obfuscation* k et un paramètre de tolérance ϵ, retourne un graphe incertain G' dans lequel est injecté le niveau minimum d'incertitude nécessaire pour se protéger de la propriété P. Celui-ci se déroule en deux temps. D'abord, l'ensemble E_C des arêtes de G' est constitué, et toutes les arêtes possibles du graphe G qui ne se retrouvent pas dans cet ensemble se voient attribuer la probabilité 0. Pour ce faire, un score d'unicité est calculé pour chaque sommet du graphe. Celui-ci évalue à quel point le sommet présente des propriétés structurelles qui le rendront plus difficile et plus long à traiter pour le fondre dans la masse. Plus un sommet possède un score d'unicité élevé, moins une arête qui lui est reliée a de chance d'être sélectionnée aléatoirement pour consister l'ensemble E_C. Par conséquent, ses liens adjacents auront plus de chance de se voir attribuer la probabilité 0. De cette façon, l'algorithme écarte les sommets problématiques pour ses performances en leur attribuant directement la plus forte distorsion possible.

Ensuite, une perturbation r_e est calculée puis appliquée pour obtenir la probabilité de chacune des arêtes de E_C. Si l'arête apparaît à la fois dans E et dans E_C, alors sa probabilité est égale à 1 - r_e. Sinon, elle est égale à r_e. Pour préserver un maximum d'utilité au graphe, cette perturbation doit rester la plus faible possible. Les auteurs proposent eux d'utiliser la distribution normale tronquée entre 0 et 1 pour obtenir les valeurs de r_e, tout en laissant la possibilité de changer en fonction des besoins spécifiques. Celle-ci permet d'obtenir en très grande majorité des valeurs proches de 0 pour r_e, qui provoqueront beaucoup plus de petites perturbations que de grosses. Chaque valeur de perturbation n'est pas attribuée par hasard sur chaque arête. L'algorithme proposé par Boldi et al. se base sur le score d'unicité des extrémités de chaque arête et attribue une plus forte perturbation à celles qui relient des sommets aux propriétés assez rares, afin de leur permettre de mieux se fondre dans la masse une fois de plus. Enfin, pour éviter que toutes les arêtes du graphe original ne soient facilement identifiables comme toutes celles du graphe incertain avec une probabilité supérieure à 0.5, une petite portion des paires de E_C est mise à part avant le calcul des probabilités et est plutôt perturbée avec une distribution uniforme, elle aussi tronquée entre 0 et 1.

En termes de performances, l'augmentation de la valeur de k et la diminution de celle de ϵ ont toutes les deux un impact négatif sur le temps d'exécution. Avec une valeur de k variant de 20 à 100 et une valeur pour ϵ de 10^{-3}, l'algorithme réussit à traiter entre 862 et 2113 arêtes par seconde. Ces scores ont été obtenus sur trois graphes différents, dont les nombres de sommets sont respectivement 226 413, 588 166 et 1 226 311. L'étude de ces trois graphes révèle deux éléments importants : la méthode est plus ou moins rapide en fonction de la topologie du graphe de base, et plus un graphe est épars, plus la méthode sera efficace. Avec les mêmes valeurs de k mais un ϵ cette fois-ci égal à 10^{-4}, les performances chutent et atteignent entre 271 et 1900 arêtes par seconde. Ces quelques données chiffrées permettent de se rendre compte de l'ordre de grandeur de temps nécessaire pour traiter des graphes de différentes tailles.

Concernant la préservation de l'utilité, celle-ci est évaluée en mesurant la différence entre les valeurs de quelques caractéristiques obtenues sur le graphe initial et la moyenne de celles obtenues sur 100 mondes possibles. Les mesures prises en considération sont le nombre d'arêtes, le degré moyen, le degré maximal, la variance du degré, la distance moyenne, le diamètre, le diamètre effectif, la moyenne harmonique des distances entre chaque paire de sommets, la distribution des distances et enfin le cœfficient de clustering. L'expérience montre que pour une petite valeur de k, plus précisément $k = 20$, l'erreur reste inférieure à 15%. Avec une grande valeur, $k = 100$, la plus grande erreur obtenue est de 70%. Il est toutefois à noter que sur des graphes épars, les résultats sont bien meilleurs, ne dépassant pas 3% même pour de grandes valeurs de k. Ceci est dû au fait que de tels graphes exigent moins de perturbations pour atteindre rapidement l'*obfuscation*.

Contrairement à l'algorithme [12], Boldi et al. proposent une méthode qui réussit à relativement bien préserver l'utilité du graphe de départ, tout en étant assez générique pour pouvoir être utilisée contre un grand nombre d'attaques possibles grâce à leur définition de l'*obfuscation*. Toutefois, comme pour [12], seules les protections contre la découverte d'identités et de liens sont mises en avant, et aucune solution n'est une nouvelle fois proposée contre la découverte d'attributs. Dans sa thèse, Nguyen [15] note également le fait que les probabilités générées par cette méthode restent toujours globalement proches de 0 pour les arêtes non-existantes dans le graphe initial, et proches de 1 pour celles qui étaient déjà présentes. Cela peut nuire à la sécurité des mondes possibles, notamment en ce qui concerne la découverte de liens. Pour se détacher de cette vérité, il faut largement augmenter le niveau de perturbation, ce qui a forcément un impact significatif sur le taux d'erreur. Il propose alors une extension de la méthode de [5] appelée *MaxVar*, dans le but de régler ce problème tout en proposant un modèle qui respecte les principes UAM, pour *Uncertain Adjacency Matrix*. Il s'agit de trois règles qui permettent de préserver une bonne utilité d'un graphe initial après application de méthodes de bruitage probabilistes. Ces trois règles sont les suivantes :

— $A_{ij} = A_{ji}$: la matrice d'adjacence du graphe incertain doit être symétrique ;
— $A_{ij} \in [0, 1]$ et $A_{ii} = 0$: le graphe ne doit pas contenir de boucles ou d'arêtes multiples entre deux sommets ;
— $\sum_{j=1}^{n} A_{ij} = d_i(G')i = 1..n$: les degrés de tous les sommets du graphe initial doivent être identiques dans le graphe incertain.

Ces restrictions permettent notamment de préserver la distribution des degrés, caractéristique très importante pour de nombreuses analyses. L'idée générale derrière *MaxVar* est de minimiser la distorsion structurelle du graphe en maximisant la variance des degrés, sans changer leur valeur, en jouant sur les probabilités des arêtes adjacentes et/ou proches. Cela est fait grâce à une méthode d'optimisation quadratique, appliquée sur un ensemble de sous-graphes obtenus à partir du graphe principal à anonymiser. Une telle manipulation permet de créer de nouvelles arêtes, selon une limite décidée par l'utilisateur, dont la probabilité pourra être utilisée pour maximiser la

variance, mais sans relier des sommets qui ne sont pas localement proches dans le graphe comme peut le faire [5]. Le tout a donc un impact moins fort sur la distorsion structurelle du graphe incertain. Nguyen confirme cette intuition avec quelques analyses comparatives sur les mêmes caractéristiques que celles mises en avant par Boldi et al., et montre que sa méthode préserve une bien meilleure utilité avec une complexité algorithmique correcte.

ii.3 Conclusion sur les méthodes d'anonymisation basées sur l'ajout de bruit

Pour conclure, nous avons pu étudier, au travers de cette seconde sous-partie, une autre famille de méthodes d'anonymisation qui ne sont pas basées sur de la compression de graphes. Celles-ci se divisent en deux groupes principaux : les techniques qui se basent sur un ajout de bruit déterministe et qui produisent, comme pour la généralisation, un seul graphe anonymisé ; et les techniques qui se basent sur un ajout de bruit probabiliste, qui visent elles à créer un modèle du graphe initial qui peut ensuite être utilisé pour créer plusieurs graphes anonymisés par insertion d'arêtes selon leur probabilité. Ces méthodes présentent souvent de très bons résultats en termes de protection vis-à-vis de deux des trois principales failles de sécurité combattues par l'anonymisation, à savoir la découverte d'identité et la découverte de lien. Cependant, peu de pistes sont explorées pour gérer la découverte d'attributs, qui concerne pourtant une partie non-négligeable des graphes issus de réseaux sociaux. Si on omet ce point, les deux approches semblent quand même se valoir. Pour ce qui concerne l'utilité du résultat de l'anonymisation, on peut noter que les méthodes par ajout de bruit possèdent quelques résultats plus intéressants que ceux obtenus par les méthodes par généralisation, notamment par rapport à la préservation de plusieurs caractéristiques structurelles en rapport avec les chemins du graphe. Elles peinent cependant plus à conserver certaines structures locales comme les communautés de sommets, bien que certaines recherches dans le domaine tentent de remédier à ce problème [15]. Par rapport aux méthodes par généralisation, on peut finalement noter un dernier grand avantage de l'ajout de bruit. Il s'agit du fait que la structure macroscopique du graphe est beaucoup moins modifiée. En effet, la plupart du temps, les graphes anonymisés possèdent relativement, voire exactement, le même nombre de sommets et d'arêtes que le graphe initial. Cela signifie que les mêmes analyses et traitements peuvent être effectués sur le résultat, comme s'il s'agissait du graphe d'origine, contrairement aux résultats des méthodes par généralisation, qui sont eux des graphes fondamentalement différents de la structure d'origine et qui agissent comme résumés.

III. La compression sans pertes

Dans les parties précédentes de cette synthèse, nous avons discuté des différentes familles de méthodes couramment utilisées pour proposer une anonymisation de graphes. Parmi elles, nous avons remarqué la grande importance des méthodes basées sur une généralisation, un concept originellement utilisé pour effectuer une compression de graphes avec pertes. Si les méthodes avec pertes constituent une partie importante des techniques de compression de graphes, elles partagent tout de même l'affiche avec des méthodes dites sans pertes. Celles-ci visent aussi à réduire la taille d'un graphe de départ, mais s'imposent la possibilité de pouvoir exactement le reconstruire à l'identique après décompression. Dans la suite de cette synthèse, nous étudierons un exemple de référence en termes de compression sans pertes, l'algorithme *WebGraph*, ainsi que deux de ses extensions. Dans un second temps, nous essayerons de déterminer quelques rôles que de telles méthodes pourraient jouer dans un cadre d'anonymisation.

i. La méthode WebGraph

Malgré la grande efficacité des méthodes de compression avec pertes, leur principe même peut parfois se révéler être leur plus grande faiblesse. En effet, dans le cadre de certaines analyses, il peut s'avérer nécessaire de souhaiter préserver l'intégralité des données et/ou de la structure des graphes. Dans ce cas, des méthodes sans pertes existent et permettent à la fois de profiter des avantages apportés par la compression, tout en n'altérant pas le jeu de données initial.

i.1 Présentation de la méthode

La référence dans le milieu des algorithmes de compression sans pertes est la méthode WebGraph, développée par Paolo Boldi et Sebastiano Vigna [7]. Comme son nom l'indique, la méthode est à l'origine spécifiée pour compresser des graphes issus du Web, où chaque sommet représente une page Internet et chaque arête un lien hypertexte. Pour imaginer leur méthode de compression, Boldi et Vigna ont identifié une propriété structurelle que possèdent les graphes issus du Web. Ils remarquent en effet qu'une très grande majorité des liens hypertextes présents dans une page Internet référencent en réalité d'autres pages présentes dans le même site ou dans le même nom de domaine. Cela est dû aux menus et autres outils de navigation, utilisés par la très grande majorité des sites Web, et principalement implémentés à l'aide de ce type de références. Vis-à-vis du graphe, cela a pour conséquence que quasiment tous les sommets, identifiés par les URL des pages, ont une bien plus grande probabilité de posséder des voisins qui possèdent le même préfixe d'URL qu'eux. Cette observation entraîne deux propriétés sur les liens de ce type de graphes, qui sont ensuite exploitées comme point de départ par la méthode WebGraph : le voisinage et la similarité. Si la liste d'adjacence d'un graphe Web est triée par ordre alphabétique, alors la plupart des sommets possèdent des voisins qui se trouvent à un indice proche du leur dans la liste, et deux entrées qui se suivent dans cette dernière ont une forte chance de partager une grande proportion de voisins en commun. Cela implique la présence régulière de redondance locale, qu'on peut donc traiter à l'aide d'une fenêtre de travail de taille réduite, et qu'il est possible de factoriser pour réduire la taille de la représentation interne du graphe. Pour rappel, la liste d'adjacence d'un graphe correspond à une liste associative dans laquelle chaque sommet du graphe est associé à la liste de tous ses voisins.

La méthode *WebGraph* consiste à modifier la liste d'adjacence de la façon suivante. Prenons en considération une partie de la liste d'adjacence d'un graphe G, représentée dans la table 1. Comme dit précédemment, les URL du graphe Web G, qui agissent comme identifiants des sommets, ont été triées par ordre alphabétique puis remplacées par des entiers qui représentent leur indice dans la liste d'adjacence triée.

Structure de départ (liste d'adjacence)		
Sommet	degré sortant	voisins
...	...	...
9	11	7, 9, 10, 11, 12, 13, 21, 22, 200, 300, 1000
10	12	7, 9, 10, 11, 20, 21, 22, 300, 301, 302, 303, 3000
11	0	
12	4	21, 22, 50, 200
...	...	...

Table 1 – *WebGraph - Étape 0 : structure de départ.*

La première modification à effectuer sur la liste consiste à retirer la redondance impliquée par

la propriété de similarité des graphes du web. En observant de plus près les listes de voisins des sommets 9, 10 et 12, on remarque un grand nombre de répétitions. Par exemple, les sommets 9 et 10 partagent les voisins 7, 9, 10, 11, 21, 22 et 300 ; et les sommets 9 et 12 les voisins 21, 22 et 200. Pour retirer cette redondance, Boldi et Vigna proposent d'utiliser le principe de listes de référence. Concrètement, cela consiste à transformer la colonne contenant les voisins du nœud en trois nouvelles colonnes. La première contient un entier indiquant la distance dans la liste d'adjacence entre le sommet qu'on prend comme référence et celui pour lequel on désire retirer les voisins redondants. Par exemple, si on décide de prendre le sommet 9 comme référence, alors on lui octroie l'entier 0. Deux options sont ensuite possibles. Soit on décide d'utiliser 9 comme référence à la fois pour 10 et pour 12, auquel cas il faudrait indiquer dans la colonne "Ref." du sommet 10 que celui-ci se trouve un indice plus bas que sa référence, et dans celle du sommet 12 que celui-ci se trouve lui trois indices plus bas. Soit on décide que 9 est la référence de 10, et que 10 est celle de 12. Dans ce cas, la colonne "Ref." du sommet 10 vaudrait 1, et celle du sommet 12 vaudrait 2. Dans la suite de l'exemple, la première option sera privilégiée. Ensuite, dans la deuxième colonne, appelée "Liste copie", on ajoute une suite de bits dont la taille correspond au nombre de voisins que possède le sommet de référence. Chaque bit correspond à un voisin et vaut 1 si le sommet courant possède lui aussi ce voisin ou 0 sinon. Pour le sommet 10, on obtient alors 11110011010. Toutes les occurrences de 1 se trouvent aux positions des sommets 7, 9, 10, 11, 21, 22 et 300 dans la liste des voisins de 9. La dernière colonne contient finalement tous les autres sommets qui ne se trouvent pas dans la liste des voisins de la référence. Comme indiqué précédemment, la propriété de localité des graphes du web permet de travailler avec une fenêtre de taille W sur la liste d'adjacence. Cela signifie qu'un sommet peut au maximum prendre comme référence un autre sommet situé à une distance inférieure à W dans la liste ($0 < Ref. < W$). On obtient, à l'issue de cette première étape, la structure intermédiaire présentée dans la table 2.

Structure avec listes de copie				
Sommet	d^+	Ref.	Liste copie	Autres nœuds
...	...	...	...	...
9	11	0		7, 9, 10, 11, 12, 13, 21, 22, 200, 300, 1000
10	12	1	11110011010	20, 301, 302, 303, 3000
11	0			
12	4	3	00000011100	50
...	...	...	...	...

Table 2 – *WebGraph - Étape 1 : ajouter les listes de copie.*

La deuxième étape de l'algorithme *WebGraph* consiste à résumer les listes de copie précédemment créées. Pour cela, on re-découpe la colonne correspondante en deux nouvelles. Dans la deuxième, on indique la taille de chaque suite de 0 ou de 1. Précédemment, deux suites avaient été créées pour les sommets 10 et 12 : 11110011010 et 00000011100. On commence alors par compter le nombre de 0 et de 1 qui s'enchaînent. Pour la première, on a alors quatre 1, deux 0, deux 1, un 0, un 1 et un 0. Pour la seconde, six 0, trois 1 et deux 0. On possède alors temporairement les découpages suivants : {4,2,2,1,1,1} et {6,3,2}. Cependant, il est pour l'instant impossible de déterminer si chaque valeur correspond à par exemple quatre 0 ou quatre 1. Pour cela, on rajoute dans chaque découpage un 0 au tout début si la suite d'origine commençait par un 0. Dans notre exemple, il faut effectuer cette manipulation sur le deuxième découpage, ce qui nous donne : {4,2,2,1,1,1} et {0,6,3,2}. Puisqu'on possède le degré de chaque sommet , il est alors possible de réduire la taille de chaque découpage en enlevant la dernière valeur. En effet, grâce à la première valeur, il est possible de savoir si la suite commençait par un 0 ou un 1. Il suffit ensuite de déplier

chaque taille suivante et il est alors évident que tous les bits qui manquent pour atteindre le degré du sommet de référence possédaient la même valeur. On peut alors réduire les découpages en blocs à : {4,2,2,1,1} et {0,6,3}. Puisque chaque bloc possède une taille forcément supérieure ou égale à 1, on peut se permettre de retirer 1 à chaque valeur à part la première (puisque celle-ci peut éventuellement posséder la valeur 0), ce qui nous donne le découpage final : {4,1,1,0,0} et {0,5,2}. On liste ces découpages en blocs dans la seconde colonne qui vient d'être ajoutée par cette étape de l'algorithme, puis dans la première on indique le nombre de blocs ajoutés pour rendre la liste auto-descriptive. On obtient la structure intermédiaire décrite dans la table 3.

Structure avec blocs de copie					
Sommet	d^+	Ref.	Nb Blocs	Blocs copie	Autres nœuds
...	...	...	...	...	...
9	11	0			7, 9, 10, 11, 12, 13, 21, 22, 200, 300, 1000
10	12	1	5	4, 1, 1, 0, 0	20, 301, 302, 303, 3000
11	0				
12	4	3	3	0, 5, 2	50
...	...	...	...	...	...

Table 3 – *WebGraph - Étape 2 : transformer les listes en blocs de copie.*

Maintenant que l'algorithme *WebGraph* s'est chargé de supprimer la redondance impliquée par la propriété de similarité, il est temps de retirer à cette étape les séquences de voisins impliquées par la propriété de localité. Pour cela, on ajoute à la structure intermédiaire trois nouvelles colonnes. Dans la première, on indique le nombre de séquences détectées dans la colonne "Autres nœuds" de la structure de l'étape précédente. Pour le sommet 9, on remarque que les voisins 9, 10, 11, 12 et 13 forment une première séquence, et les sommets 21 et 22 une seconde. Pour le sommet 10, on en détecte une formée par les sommets 301, 302 et 303. Dans la deuxième nouvelle colonne, on va représenter les valeurs des bornes gauches des intervalles qui résument chaque séquence. Par exemple, la première séquence du sommet 9 correspond à l'intervalle [9,13] et la seconde [21,22]. Pour le sommet 10, on a [301,303]. On ne va alors représenter que les bornes gauches de chaque intervalle, 9, 21 et 301, valeurs qui seront ensuite complétées dans la troisième colonne par les tailles des intervalles. Toujours dans l'optique de réduire au maximum le nombre de bits qui seront nécessaires pour représenter à la fin les entiers de la structure, les valeurs de ces bornes gauches ne sont pas représentées telles quelles. Pour les premiers intervalles de chaque sommet, on représente leur borne gauche par la différence entre leur valeur et l'indice du sommet. On remarque alors un premier risque de se retrouver avec des valeurs négatives dans la structure, ce que *WebGraph* essaye d'éviter. En effet, même si ce n'est pas le cas ici (9 - 9 = 0 et 301 - 10 = 291), si une borne gauche possédait une valeur inférieure à l'indice du sommet, cette différence serait négative. Pour contrer ce problème, Boldi et Vigna proposent d'encoder différemment les différences entre une première valeur d'une colonne et un indice de sommet. Si la différence est supérieure ou égale à 0, on la multiplie par 2. Sinon, on multiplie sa valeur absolue par 2, puis on retire 1 :

$$v(x) = \begin{cases} 2x & if x \geq 0 \\ 2x - 1 & if x < 0 \end{cases}$$

Cela signifie que nos premières bornes gauches ne valent plus 0 et 291, mais 0×2 et 291×2, soit 0 et 582. Pour les intervalles qui suivent, on encode la valeur de leur bornes de gauche en leur retirant successivement la valeur de la dernière borne de droite puis 2. Pour [21,22] du sommet 9,

on ajoute alors la valeur 21 - 13 - 2 = 6. *WebGraph* prend en paramètre un seuil d'intervalle L_{min}, dont la valeur peut varier entre 2 et ∞, et qui permet de préciser la taille minimale des intervalles qu'on souhaite prendre en considération. Puisqu'on sait qu'un intervalle possède au minimum une taille de L_{min}, il est possible de réduire les valeurs de la troisième nouvelle colonne en retirant à chaque fois cette valeur. Ainsi, pour les intervalles du sommet 9, on peut inscrire les valeurs 3 et 0 plutôt que 5 et 2 avec un L_{min} = 2. On obtient alors la structure intermédiaire présentée dans la table 4.

Structure sans séquence								
S	d^+	Ref.	Nb Blocs	Blocs copie	Nb Seq	BG	Tailles	Autres nœuds
...	...	...	...	...	...	...	...	...
9	11	0			2	0, 6	3, 0	7, 200, 300, 1000
10	12	1	5	4, 1, 1, 0, 0	1	582	1	20, 3000
11	0							
12	4	3	3	0, 5, 2	0			50
...	...	...	...	...	...	...	...	...

Table 4 – *WebGraph - Étape 3 : coder les séquences (L_{min} = 2).*

Enfin, la dernière modification sur la structure que propose *WebGraph* consiste à réduire la valeur des entiers présents dans la colonne "Autres nœuds". Pour cela, on utilise la même technique que précédemment avec les bornes gauches des intervalles. Cela permet de ne représenter que les écarts entre les valeurs. Si on note V_j^i le sommet à l'indice j de la colonne "Autres nœuds" du sommet i, alors la valeur d'écart vaut $V_j^i - V_{j-1}^i - 1$. Si V_j^i correspond au premier sommet de la colonne, alors la différence doit être faite avec l'indice du sommet : V_j^i - i. Comme précédemment, puisqu'il s'agit d'une différence entre une valeur de colonne et un indice de ligne, il faut penser à utiliser la fonction $v(x)$ sur le résultat. La colonne "Autres nœuds" du sommet 9 devient par exemple {|7-9| $\times$ 2 - 1, 200 - 7 - 1, 300 - 200 - 1, 1000 - 300 - 1}, soit {3, 192, 99, 699}. Pour le sommet 10, {|20-10| $\times$ 2, 3000 - 20 - 1}, soit {20, 2979}. On obtient ainsi la structure de données finale de la figure 5.

Structure finale								
S	d^+	Ref.	Nb Blocs	Blocs copie	Nb Seq	BG	Tailles	Autres nœuds
...	...	...	...	...	...	...	...	...
9	11	0			2	0, 6	3, 0	3, 192, 99, 699
10	12	1	5	4, 1, 1, 0, 0	1	582	1	20, 2979
11	0							
12	4	3	3	0, 5, 2	0			50
...	...	...	...	...	...	...	...	...

Table 5 – *WebGraph - Étape 4 : ne garder que les écarts.*

On remarque que cette structure est auto-descriptive. Cela permet de l'encoder facilement comme une suite d'entiers en respectant un certain nombre de consignes. Les points clés de la phase d'encodage sont :

— toujours indiquer le degré de chaque ligne ;
— si "d^+" > 0, ajouter la valeur de "Ref." ;
— si "Ref." > 0, ajouter la valeur de "Nb Blocs" puis celles contenues dans "Blocs copie" ;

— si la somme de la taille des blocs qui représentent des 1 dans "Blocs copie" est supérieure ou égale au degré, alors passer au sommet suivant.
— sinon, regarder la valeur de L_{min} ;
— si $L_{min} = \infty$, alors ajouter les valeurs contenues dans "Autres nœuds" ;
— sinon, ajouter le nombre de séquences, puis chaque borne gauche suivie de sa taille (BG1 Taille1, BG2 Taille2, ..., BGn TailleN) et enfin les autres nœuds ;

Résultat final :
11 0 2 0 3 6 0 3 192 99 699
12 1 5 4 1 1 0 0 1 582 1 20 2979
0
4 3 3 0 5 2 0 50

Avec l'algorithme *WebGraph*, on obtient une représentation des 4 sommets 9, 10, 11 et 12 de G en une suite de 33 entiers. Il est ensuite possible de les encoder en utilisant un format de compression des entiers, comme Golomb ou le γ-code par exemple. La structure finale de *WebGraph* peut être utilisée avec un système de *lazy iteration*. Cela signifie que si on le souhaite, on peut travailler sur la structure compressée comme s'il s'agissait de la liste d'adjacence initiale. Dans ce cas, il suffit de juste reconstruire la liste des voisins de base à partir des informations de la structure pour le sommet qui nous intéresse uniquement au moment où on en a besoin. Si pour ce faire il est nécessaire de reconstruire la liste des voisins d'un sommet de référence, alors on le fait récursivement jusqu'à tomber sur une valeur de "Ref." égale à 0. Au moment de la construction de la structure, *WebGraph* propose la possibilité de fixer une limite sur le nombre de références récursives possibles, afin d'offrir des meilleures performances d'utilisation par *lazy iteration*. Cela se fait bien évidemment au prix d'un moins fort taux de compression, tout comme la variation de la taille de fenêtre de travail W.

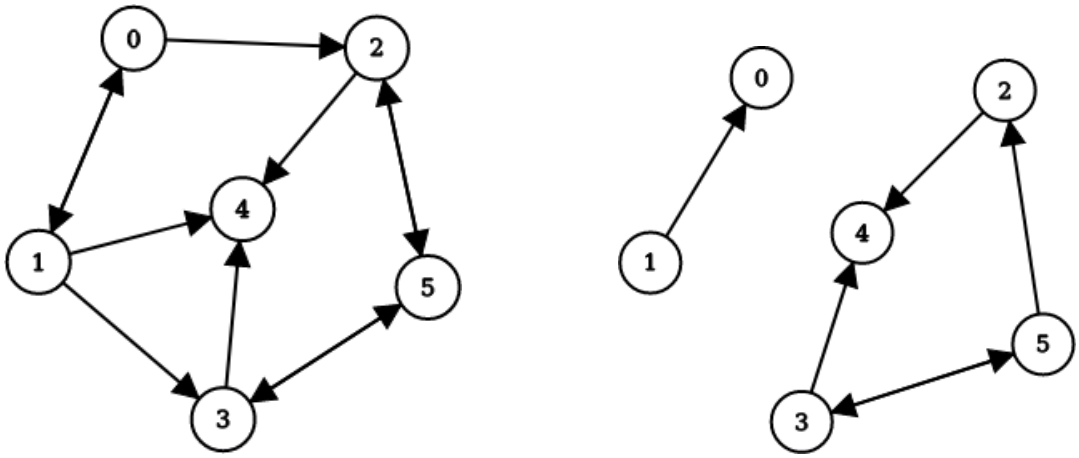
Grâce à cette technique, Boldi et Vigna réussissent à atteindre d'excellents scores de compression, de l'ordre d'une trentaine de bits par sommet et de moins de 4 bits par arête. À titre de comparaison, les méthodes similaires de l'époque mentionnées dans l'article, celles de Randall et al. [17] et de Raghavan et Garcia-Molina [16], présentent respectivement des scores de 5.61 et de 5.07 bits par lien. Même avec des limites sur le temps d'exécution, posées à travers le nombre maximum de références récursives possibles, l'algorithme *WebGraph* réussit à surpasser les résultats de ces deux précédentes méthodes. L'algorithme présente cependant quelques faiblesses et inconvénients. Par exemple, aucun moyen efficace n'est proposé pour réduire correctement la taille nécessaire pour stocker les URL, ce qui est problématique lorsque celles-ci sont longues. Mais sa plus grande faiblesse réside dans le fait que la méthode est spécifiquement créée et adaptée pour des graphes Web, et qu'elle est ainsi difficilement généralisable telle quelle pour les autres types de graphes.

i.2 Généraliser sur tout type de graphe : l'approche d'Apostolico et Drovandi

Pour résoudre ce dernier problème, Alberto Apostolico et Guido Drovandi [1] ont ensuite travaillé sur un nouveau moyen d'ordonner les sommets dans la liste d'adjacence avant d'appliquer la méthode [7], applicable cette fois-ci sur tous les types de graphes. Pour cela, ils proposent d'utiliser le fameux algorithme de parcours en largeur d'un graphe, en anglais Breadth-First Search ou BFS. L'analyse se base uniquement sur la topologie du graphe, ce qui permet la prise en charge de réseaux qui n'utilisent pas d'URL. Leur algorithme étend ensuite *WebGraph* de la manière suivante.

Tout d'abord, un parcours en largeur du graphe est effectué. Celui-ci permet d'ordonner la liste d'adjacence. Pour rappel, BFS est une méthode de parcours de graphe utilisant une structure

de données de type file. À l'étape d'initialisation, un premier sommet est ajouté. On retire ensuite à chaque étape le premier élément de la file pour le traiter, jusqu'à ce que celle-ci soit vide. Ce traitement consiste à ajouter à la fin de la file tous les voisins du sommet qui n'ont pas encore été ajoutés à un moment ou à un autre dans la structure. L'ordre d'ajout de chaque sommet dans la file définit l'ordre BFS. Ainsi, le premier sommet du graphe à avoir été ajouté passe à l'indice 0 de la liste d'adjacence, le deuxième à l'indice 1, et ainsi de suite jusqu'à atteindre l'indice n-1 (avec n le nombre de sommets du graphe). À chaque fois qu'un sommet est ajouté, Apostolico et Drovandi proposent de simplifier le graphe de départ, pour commencer à réduire la taille de la liste d'adjacence. Pour cela, ils introduisent le principe de *traversal list*. Dans celle-ci, on indique à chaque traitement de BFS le nombre de sommets qui sont ajoutés à la file. Avec cette information, on peut supprimer un ensemble d'arêtes du graphe, dont leur existence est sous-entendue par les indices attribués aux différents sommets. Ce phénomène est illustré sur la figure 9. Grâce à la *traversal list*, on sait que le traitement par BFS du sommet d'indice 0 a entraîné l'ajout dans la file de deux autres sommets (T à l'indice 0 vaut 2). Cela signifie automatiquement que les sommets aux indices 1 et 2 sont des voisins de 0, il n'y a donc pas besoin de les représenter dans sa liste de voisins. On sait ensuite que le traitement de 1 a ajouté à la file deux autres de ses voisins. Cela signifie donc forcément que les sommets d'indice 3 et 4 sont des voisins de 1. On peut donc retirer l'arc qui les sépare. En revanche, le traitement de 1 n'a pas entraîné l'ajout de 0 à la file, puisque celui-ci était déjà traité. Ce voisin doit donc toujours figurer dans la liste d'adjacence. Ce raisonnement est appliqué lors du traitement de chaque sommet.



(a) *Graphe de départ indicé avec BFS*

(b) *Graphe simplifié à l'aide de la traversal list (T = {2,2,1,0,0,0})*

Figure 9 – *Ordonner la liste d'adjacence grâce à BFS (exemple tiré de [1]).*

La première étape de l'algorithme [1] permet donc de faire apparaître une propriété de localité dans la liste d'adjacence simplifiée, similaire à celle exploitée par *WebGraph*. La deuxième phase consiste ensuite à faire apparaître de la redondance, conséquence précédemment de la propriété de similarité. Pour cela, la liste des voisins de chaque sommet est encodée en représentant les écarts entre chaque valeur, un peu comme le faisait la dernière étape de la construction de la structure de *WebGraph*. Cependant, un plus grand nombre de formules de calcul de l'écart sont cette fois-ci présentées, en fonction des cas à traiter. Chacune est représentée par un symbole différent, qui préfixe la valeur d'écart obtenue. La redondance entre les couples symbole-écart est ensuite quantifiée de façon à obtenir une nouvelle fois une structure auto-descriptive, présentée

comme une succession d'entiers et de symboles (caractères). Les entiers sont compressés à l'aide du format π-code, une variante du γ-code proposé par *WebGraph*, et les symboles sont eux compressés avec le codage d'Huffman. Le but de ces deux méthodes est de représenter avec le moins de bits possibles les valeurs qui ont beaucoup d'occurrences et inversement. À titre de comparaison, utiliser l'ordre fourni par BFS plutôt que l'ordre alphabétique dans l'algorithme *WebGraph* augmente en moyenne le nombre de bits par liens nécessaires de 2.5%. Toutefois, avec la méthode [1] complète, c'est-à-dire en utilisant à la fois l'ordre BFS et le codage particulier qu'ils proposent, il est possible d'atteindre de meilleurs résultats que Boldi et Vigna, avec un taux de compression de 1.87 bits par lien. Dans tous les cas, l'ordre BFS seul permet tout de même d'étendre la méthode WebGraph pour prendre en charge tous les types de graphes, au prix d'un taux de compression légèrement plus faible.

i.3 Se spécialiser pour les réseaux sociaux : l'extension de Boldi et Vigna

Quelques années après la spécification du framework *WebGraph* [7], Boldi et Vigna présentent une nouvelle méthode, spécialisée cette fois-ci sur la compression de graphes issus de réseaux sociaux [6]. La méthode de base est identique à celle mise en place pour les graphes du web, et leur contribution consiste, comme pour [1], en un moyen de déterminer le meilleur ordre de sommets à utiliser pour la liste d'adjacence, dans le cas spécifique du traitement de graphes sociaux. En étudiant les différentes méthodes pour définir des ordres de sommets, dont celle proposée par [1], Boldi et Vigna font l'observation que plus l'ordre permet de rapprocher les sommets d'un même groupe de voisinage entre eux, plus celui-ci proposera de bons scores de compression par la suite. Ils appellent ces groupes de voisinage des hôtes, qui correspondent intuitivement à la notion traduite par les domaines des sites Internet dans le contexte d'un graphe web. Leur proposition est alors de transformer le problème d'ordonnancement de sommets en un problème de clusterisation, afin de capter ces groupes au sein du graphe. Ils choisissent pour cela d'utiliser un algorithme par propagation de labels, qui correspond bien aux contraintes imposées par le contexte. En effet, les algorithmes de ce type ne nécessitent pas de connaître à l'avance le nombre de groupes à former et sont de complexité linéaire par rapport au nombre de liens du graphe, ce qui est adapté aux graphes de réseaux sociaux, qui sont souvent de grande taille. Le principe général de la propagation de labels est d'associer à chaque sommet du graphe une étiquette qui représente le groupe dans lequel il se trouve. Au départ, chaque sommet se trouve dans un groupe différent. Plusieurs lectures du graphe sont ensuite effectuées et les étiquettes sont mises à jour en appliquant un certain nombre de règles, qui peuvent varier selon l'algorithme précis de propagation de labels. Dans le contexte de leur méthode, [6] proposent une règle qui implique de choisir pour chaque sommet l'étiquette d'un de ses voisins qui maximise une formule qui met en relation trois paramètres : le nombre de voisins du sommet qui possèdent cette étiquette, le nombre de sommets du graphe qui la possèdent et enfin un paramètre de densité à fixer expérimentalement à l'avance. Le principe de l'algorithme LLP [6], pour *Layered Label Propagation*, est ensuite de lancer successivement plusieurs itérations de la propagation de labels, avec à chaque fois une valeur de densité différente. Chaque étape produit donc une clusterisation du graphe et les sommets qui appartiennent au même groupe sont placés et ordonnés à la suite dans la liste d'adjacence. Une fois cela fait, les différents ordres créés à chaque étape sont étudiés conjointement pour construire un ordre final retourné par l'algorithme. Celui-ci est finalement utilisé à la place de l'ordre alphabétique dans l'algorithme [7]. Après comparaison des taux de compression, cette méthode s'avère en moyenne 25% plus efficace que celle proposée par [1] sur les jeux de données testés par Boldi et Vigna. Ils observent également que cet écart de performance tend à se creuser avec l'augmentation de la taille des graphes en entrée, ce qui est signe d'une

moins forte complexité.

ii. L'apport possible

Dans la sous-section précédente ont été présentées trois méthodes de compression de graphes sans pertes. De cette étude, il est possible de conclure qu'en tant que telles, ces techniques ne produisent en aucun cas de l'anonymisation, contrairement à ce que pouvaient naturellement proposer les méthodes avec pertes contre les découvertes d'identité et de lien. Ceci est une conséquence directe de l'objectif principal de la compression sans pertes : pouvoir reconstruire le graphe d'origine à l'identique après décompression. Et de fait, cela implique que le graphe compressé ne possède pas plus de propriétés de sécurité que son homologue décompressé. Mais les méthodes de compression sans pertes n'ont-elles pour autant aucun rôle à jouer dans l'anonymisation ? Il est en réalité possible que si.

Comme il l'a très régulièrement été répété au travers de la première section, les méthodes d'anonymisation partagent souvent un point commun : leur complexité algorithmique est assez élevée. Cela peut impacter de manière plus ou moins forte leur utilité sur de grands graphes. Or, comme nous l'avons précisé dans l'introduction de cette synthèse, les graphes issus d'environnements sociaux, qui sont au cœur de la problématique d'anonymisation, tendent à toujours plus grossir avec le temps et de très nombreuses études à leur sujet aimeraient pouvoir tirer profit de cela. Comme nous l'avons vu précédemment, les méthodes de compression sans pertes dérivées de *WebGraph* possèdent d'excellents scores de compression. Un rôle que pourrait jouer la compression sans pertes pourrait alors être de constituer une étape de pré-traitement, qui permettrait de réduire de façon importante la taille du graphe à anonymiser. Cela permettrait de faciliter à la fois le traitement et la publication de données de taille significative. De plus, puisque ces méthodes ne suppriment aucune donnée du graphe d'origine, elles ne provoquent aucune distorsion qui pourrait encore plus réduire l'utilité préservée par certaines anonymisations.

Cependant, il existe un problème de taille à gérer pour pouvoir faire cela. En effet, le résultat des méthodes dérivées de *WebGraph* est une structure loin d'être similaire à la simple liste d'adjacence du graphe d'origine. Plusieurs idées de solutions pour contourner ce problème peuvent être discutées. Leur pertinence n'a nullement été prouvée expérimentalement, et celles-ci constituent essentiellement des ouvertures possibles proposées par cette synthèse. Dans un premier temps, si on souhaite tout de même pouvoir utiliser les méthodes d'anonymisation présentées dans la première section, une idée pourrait être de leur ajouter un système similaire à de la *lazy evaluation*. Cette technique, qui est à l'origine du système de *lazy iteration* de *WebGraph*, et qui est beaucoup utilisée dans plusieurs langages de programmation fonctionnelle, consiste à ne calculer une valeur qu'au moment où on en a besoin. Dans notre contexte, un système similaire pourrait être utilisé pour travailler au maximum avec la structure compressée, et ne décompresser que temporairement seulement les parties du graphe qui nous intéressent à un instant donné de l'algorithme d'anonymisation. Une deuxième idée pourrait être de re-spécifier ces méthodes (si possible) pour leur permettre de travailler sur la structure compressée. Enfin, une dernière idée consiste simplement à créer de nouvelles extensions de l'algorithme *Webgraph* ou de ses variantes, modifiées pour créer de l'anonymat. Pour de l'anonymisation, où les graphes sont en général issus de réseaux sociaux, étendre une des variantes semble d'ailleurs plus judicieux. Il faudrait alors se détacher de l'aspect sans pertes de ces méthodes de compression.

Sergey Kirgizov propose par exemple de conserver le même principe général que l'algorithme *WebGraph*, mais en utilisant plutôt un système d'encodage des entiers non-bijectif (surjectif même). Ainsi, lors de la décompression du graphe anonymisé pour pouvoir travailler dessus, un même code pourrait correspondre à plusieurs entiers différents, ce qui créerait de l'incertitude de la

même manière que les méthodes par ajout de bruit probabiliste. Cependant, cette idée risque de provoquer une bien trop grosse distorsion du graphe de base. En effet, puisque tous les entiers seraient soumis à ce système d'encodage, alors toutes les colonnes de la structure seraient impactées, ce qui modifierait à la fois le degré, les blocs, les intervalles, les autres voisins, etc. Il se peut même que la structure elle-même devienne incohérente et ne puisse plus être correctement décompressée.

Pour modifier la structure sans prendre ce genre de risques, une solution pourrait être d'uniquement modifier la colonne "Autres nœuds". Deux avantages se démarquent en se concentrant uniquement sur cette dernière. Déjà, il s'agit de la colonne de la structure avec la sémantique la moins forte de toutes pour la compression et la décompression. Il suffit que le nombre d'entiers qui la composent reste identique après ajout de bruit pour ne pas rendre le tout incohérent. Ensuite, les sommets de cette colonne représentent les liens rares, ceux qui ne sont soumis ni à la propriété de localité, ni à celle de similarité. En les modifiant, on réussirait ainsi à tout de même provoquer une distorsion dans la topologie du graphe, sans toucher non plus aux structures locales denses. Cependant, une telle idée implique le fait que les degrés des différents sommets ne changent pas. Or, comme nous l'avons vu précédemment, les attaques qui utilisent cette propriété structurelle sont assez courantes, ce qui laisse à douter du niveau de sécurité offert par une telle méthode. Une piste pour résoudre ce problème pourrait alors être de supprimer avec une certaine probabilité certains voisins se trouvant dans la colonne "Autres nœuds", en veillant bien dans ce cas à modifier la valeur du degré pour ne pas rendre la structure compressée incohérente.

Dans tous les cas, on peut conclure sur l'utilisation des méthodes de compression sans pertes dans un contexte d'anonymisation. Ces méthodes, ou tout du moins celles dérivées de *WebGraph*, ont tendance à être structurée par l'enchaînement de deux phases : une première qui détermine une façon optimale d'ordonner et présenter la liste d'adjacence, de façon à faire apparaître de la redondance et des intervalles de valeurs sur une même ligne et d'une ligne à l'autre ; et une seconde qui se charge d'encoder cette redondance puis le reste de la structure en utilisant un ou plusieurs formats dans le but de minimiser le nombre de bits nécessaires. Si on souhaite transformer ces méthodes pour proposer de l'anonymisation, il est nécessaire de provoquer une perturbation à une ou plusieurs étapes de la compression, ce qui leur fera inexorablement perdre leur facteur sans pertes. La deuxième phase semble être le meilleur moment pour cela. En effet, il est intéressant de conserver intacte la phase de recherche de l'ordre optimal pour la liste d'adjacence, car les propriétés de localité et de similarité semblent être de bons candidats pour identifier les zones de sommets à ne pas ou peu modifier afin de ne pas trop distordre la topologie générale du graphe d'origine. Ainsi, de fait, l'utilité générale pourrait être mieux préservée. Par conséquent, cette deuxième phase semble être un choix plus judicieux.

IV. Conclusion

Pour conclure, nous avons étudié au cours de cette synthèse deux familles de méthodes d'anonymisation de graphes sociaux. La première est constituée de techniques basées sur un outil originellement crée pour la compression de graphes avec pertes, la généralisation. La seconde rassemble les méthodes basées sur de l'injection de bruit. Nous avons vu que l'anonymisation consiste à modifier les données d'un graphe, dans le but de maximiser le niveau de sécurité des informations face à trois principales failles, tout en préservant le plus possible l'utilité du résultat pour qu'il puisse être utilisable à la place du graphe d'origine dans le cadre d'études. La première attaque combattue par l'anonymisation, la découverte d'identité, consiste à associer un individu du monde réel au sommet qui le représente dans le graphe. La deuxième, la découverte d'attributs, consiste à découvrir des données privées et potentiellement sensibles sur l'individu à partir des

données du graphe. La dernière, la découverte de lien, consiste à déterminer que deux entités du monde réel sont liées. Pour contrer ces trois failles, les méthodes d'anonymisation essayent d'atteindre un seuil de k-anonymat, qui garantit l'impossibilité de discerner moins de k sommet à partir d'une connaissance a priori donnée sur le graphe. Après avoir passé en revue quelques méthodes d'anonymisation pour comprendre les enjeux et les défis de ces méthodes, nous nous sommes intéressés aux méthodes de compression de graphes sans pertes, afin de déterminer le rôle que celles-ci pourraient jouer pour effectuer de l'anonymisation. Nous avons étudié le cas particulier de la méthode *WebGraph*, ainsi que deux de ses extensions. Finalement, nous avons conclu que les méthodes d'anonymisation par compression sans pertes ne constituent pas en tant que telles des méthodes d'anonymisation. Cependant, elles peuvent tout de même avoir un rôle à jouer dans ce milieu, que ce soit en tant que phase de pré-traitement pour pouvoir anonymiser de grands graphes en réduisant la taille des données en entrée ; ou tout simplement comme méthodes de base à étendre en ajoutant judicieusement de la perte d'informations, de sorte à atteindre ce nouvel objectif. Dans ce dernier cas, leur capacité à faire émerger les structures locales redondantes au sein des graphes pourrait leur permettre de préserver une bonne utilité au graphe de base, comme souvent avec les méthodes qui évitent au maximum de grosses distorsions. Ces rôles restent cependant pour l'instant toujours hypothétiques, et un travail pour vérifier leur faisabilité pourrait constituer une bonne extension à cette synthèse.

Références

[1] A. Apostolico and G. Drovandi. Graph compression by bfs. *Algorithms*, 2(3) :1031–1044, 2009.

[2] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x ? anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th international conference on World Wide Web*, pages 181–190, 2007.

[3] V. Batagelj and M. Zaveršnik. Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification*, 5(2) :129–145, 2011.

[4] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics : theory and experiment*, 2008(10) :P10008, 2008.

[5] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa. Injecting uncertainty in graphs for identity obfuscation. *arXiv preprint arXiv :1208.4145*, 2012.

[6] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation : A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World wide web*, pages 587–596, 2011.

[7] P. Boldi and S. Vigna. The webgraph framework i : compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602, 2004.

[8] F. Bonchi, A. Gionis, and T. Tassa. Identity obfuscation in graphs through the information theoretic lens. *Information Sciences*, 275 :232–256, 2014.

[9] A. Campan and T. M. Truta. A clustering approach for data and structural anonymity in social networks, 2008.

[10] J. Casas-Roma and F. Rousseau. Community-preserving generalization of social networks. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1465–1472. IEEE, 2015.

[11] A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6) :066111, 2004.

[12] T. Feder, S. U. Nabar, and E. Terzi. Anonymizing graphs. *arXiv preprint arXiv :0810.5578*, 2008.

[13] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis. Resisting structural re-identification in anonymized social networks. *Proceedings of the VLDB Endowment*, 1(1) :102–114, 2008.

[14] Y. Liu, T. Safavi, A. Dighe, and D. Koutra. Graph summarization methods and applications : A survey. *ACM Computing Surveys (CSUR)*, 51(3) :1–34, 2018.

[15] H.-H. Nguyen. *Social Graph Anonymization*. PhD thesis, 2016.

[16] S. Raghavan and H. Garcia-Molina. Representing web graphs. In *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405)*, pages 405–416. IEEE, 2003.

[17] K. H. Randall, R. Stata, R. G. Wickremesinghe, and J. L. Wiener. The link database : Fast access to graphs of the web. In *Proceedings DCC 2002. Data Compression Conference*, pages 122–131. IEEE, 2002.

[18] M. E. Skarkala, M. Maragoudakis, S. Gritzalis, L. Mitrou, H. Toivonen, and P. Moen. Privacy preservation by k-anonymization of weighted social networks. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 423–428. IEEE, 2012.

[19] K. Stokes et al. On some clustering approaches for graphs. In *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*, pages 409–415. IEEE, 2011.

[20] X. Ying and X. Wu. Randomizing social networks : a spectrum preserving approach. In *proceedings of the 2008 SIAM International Conference on Data Mining*, pages 739–750. SIAM, 2008.