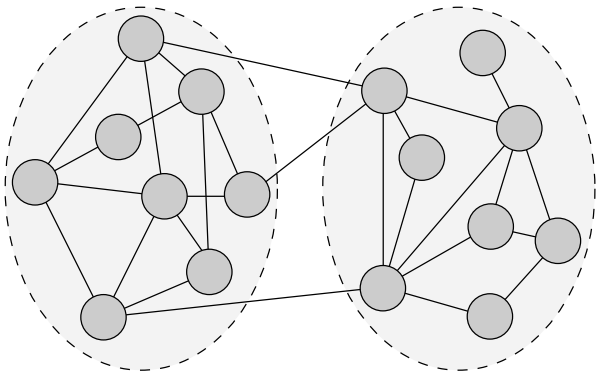
Rapport de DEA

Algorithmique des grands réseaux d'interactions : détection de structures de communautés

15 juillet 2004

Stage effectué au LIAFA (CNRS - Paris 7)
dans le cadre du DEA d'Algorithmique (ENS Paris - Paris 7)



Encadré par :
Matthieu Latapy
latapy@liafa.jussieu.fr

Soutenu par :
Pascal Pons
pascal.pons@ens.fr

Table des matières

1 Introduction

1.1 Les grands réseaux d'interactions

Les récentes avancées dans le domaine des systèmes complexes ont fait ressortir le rôle central que jouent les grands réseaux d'interactions dans la dynamique de nombreux phénomènes courants. Ces réseaux peuvent être modélisés par des graphes dont les sommets représentent les acteurs du phénomène et les liens représentent les interactions entre eux. Nous pouvons citer plusieurs types de tels réseaux : les réseaux sociaux (réseaux de connaissances, réseaux de collaboration scientifique, réseaux des appels téléphoniques), les réseaux biologiques (réseaux métaboliques, réseaux de neurones, réseaux trophiques), les réseaux d'infrastructure (réseaux routiers, réseaux de distribution d'électricité, réseau physique de l'Internet), les réseaux d'information (réseau des pages Web, réseaux de citations d'articles), etc... Nous renvoyons aux travaux de synthèse [1, 2, 3, 4, 5] pour une bibliographie complète du domaine.

Tous ces graphes, bien que d'origines différentes, possèdent des propriétés statistiques et structurelles communes. Leur distribution de degrés suit en général une loi de puissance, et leur degré moyen est relativement faible, ce qui donne des graphes peu denses. De plus, la distance moyenne entre sommets est faible (effet "petit monde"). Enfin, les graphes ont un coefficient de clustering élevé (ce coefficient représente la probabilité pour que deux voisins d'un même sommet soient eux mêmes connectés). Cette dernière propriété traduit l'existence de zones où la densité locale de liens est élevée par rapport à la densité globale.

L'étude de ces grands réseaux d'interactions soulève de nombreux problèmes algorithmique qui deviennent rapidement de véritables défis vues les tailles des graphes rencontrés (jusqu'à quelques milliards de sommets pour le graphe du Web). Pour s'adapter à ces tailles, il est nécessaire de développer une algorithmique spécifique aux grands réseaux d'interactions pouvant tirer parti de leurs propriétés communes. Nous allons proposer dans cette optique un algorithme de détection de structure de communautés.

1.2 Présentation du problème de détection de structures de communautés

L'existence dans les réseaux d'interactions de zones plus densément connectées que d'autres découle souvent de la présence dans le graphe d'une structure de communautés. Cette notion correspond intuitivement à l'existence de groupes de sommets plus fortement connectés entre eux que vers les autres sommets. Ce type de structure intervient dans de nombreux réseaux d'interactions et apporte de l'information sur leur organisation. Les communautés peuvent avoir des interprétations différentes suivant le type de réseau considéré. Ainsi, le terme de communauté a d'abord été introduit pour les réseaux sociaux selon son interprétation naturelle. Dans les autres domaines, cette même notion peut aussi être nommée "modularité". Pour les réseaux métaboliques, les communautés correspondent par exemple à des fonctions biologiques de la cellule [6]. Pour les réseaux d'information, elles correspondent à des thématique. Par exemple les pages Web traitant d'un même sujet se référencent mutuellement et la détection de communautés dans le graphe du Web est une piste envisagée pour améliorer les moteurs de recherche [7]. La détection de communautés est donc un outil important pour la compréhension de la structure et du fonctionnement des grands réseaux d'interactions.

La notion de communauté dans un graphe est cependant difficile à définir formellement, il n'existe pas à ce jour de définition satisfaisante. De nombreuses définitions ont été introduites pour l'étude des réseaux sociaux [1]. Nous donnons brièvement ici les notions principales : la définition la plus stricte correspond aux *cliques* maximales du graphe. On peut considérer une

version plus faible en utilisant des *n-cliques* (sous-ensemble de sommets se trouvant à distance au plus n les uns des autres). Viennent ensuite les définitions des *LS sets* et des *Lambda sets* : un LS set est un sous-ensemble S de sommets tel que tout sous-ensemble propre de S a plus de liens vers son complément dans S que vers l'extérieur de S. Un Lambda set est un sous-ensemble S de sommets dans lequel il existe toujours plus de chemins indépendants entre une paire de sommets de S que entre un sommet de S et un sommet extérieur à S.

Ces définitions formelles sont trop restrictives et ne se prêtent pas à des algorithmes efficaces de détection de communautés. C'est pourquoi toutes les approches récentes ont utilisé une approche moins formelle et plus intuitive de la notion de communauté. Une communauté est alors vue comme un ensemble de sommets dont la densité de connexions internes est plus forte que la densité de connexions vers l'extérieur. Par exemple, [8] définit deux notions de communautés : les communautés au sens faible sont des sous-ensembles de sommets ayant globalement plus de liens internes que de liens externes et les communautés au sens fort vérifient cette même propriété pour chacun des sommets.

Une autre approche consiste à quantifier la notion intuitive de communautés en attribuant à tout sous-ensemble de sommets un score caractérisant la cohésion de cette hypothétique communauté. Le but de la détection de communautés est alors de trouver la meilleure partition $\mathcal{C} = C_1, C_2, \ldots, C_k$ de l'ensemble des sommets pour maximiser le score des communautés C_i ainsi définies, tout en minimisant le nombre k de communautés. Nous utiliserons cette dernière approche dans notre étude en considérant la notion de modularité introduite dans [9] et qui sera présentée en 2.2.2. Nous regrouperons les sommets par communautés en nous basant sur l'étude de la dynamique des marches aléatoires dans le graphe.

2 Etat de l'art

Nous présentons dans cette section un panorama des différentes approches qui ont été envisagées pour extraire la structure de communautés d'un graphe. Les premiers algorithmes s'apparentant à cette problématique ont été développés pour les problèmes de partitionnement de graphe et pour les problèmes de clustering hiérarchique. Nous verrons ensuite les algorithmes récents qui sont spécialement développés pour la détection de communautés. Finalement, nous introduirons les grandes lignes de notre approche.

2.1 Les approches classiques

2.1.1 Le partitionnement de graphe

Le but du partitionnement de graphe est de grouper les sommets d'un graphe en un nombre prédéterminé de parties (de tailles elles aussi prédéterminées) tout en minimisant le nombre d'arêtes tombant entre les différents groupes. Cette approche ne convient pas totalement à ce que l'on cherche car elle a l'inconvénient de requérir une connaissance préalable du nombre de communautés recherchées ainsi que de leur taille. Nous citons les deux méthodes ayant eu le plus de succès.

La méthode de bissection spectrale [10, 11]. Elle consiste à calculer le vecteur propre correspondant à la plus petite valeur propre non nulle de la matrice Laplacienne du graphe $L = D - A$ (selon les notations qui seront introduites en 3.1). Le graphe est alors séparé en deux parties en fonction du signe de leur composante selon ce vecteur propre. Cette méthode

fonctionne en temps $\mathcal{O}(n^3)$ et obtient de bons résultats lorsque le graphe possède effectivement deux grandes communautés de tailles similaires, ce qui n'est qu'un cas particulier dans l'optique de détection de communautés.

La méthode de Kernighan et Lin [12]. Il s'agit d'un algorithme de bissection visant à trouver la coupe du graphe minimisant le nombre d'arêtes tombant entre les deux groupes. L'algorithme nécessite en paramètre la taille des communautés à détecter, une coupe de la bonne taille est choisie aléatoirement comme point de départ et utilise une heuristique gloutonne. La bissection est améliorée itérativement en faisant des échanges de sommets entre les communautés. À chaque étape on échange les deux sommets procurant la meilleure réduction du nombre d'arêtes externes, avec la condition imposée de ne jamais changer deux fois un même sommet. Il y a donc exactement $\frac{n}{2}$ étapes de calcul et la meilleure partition rencontrée au cours du déroulement de l'algorithme est retenue. La complexité du pire cas est en $\mathcal{O}(n^3)$.

2.1.2 Les méthodes de clustering hiérarchique

Cette méthode a été introduite pour analyser des données [13, 14]. Elle s'adapte parfaitement aux graphes : son but est alors de grouper les sommets en sous-ensembles (qui représenteront les communautés) de telle sorte que chaque sommet soit groupé avec d'autres sommets similaires. Pour cela, il est nécessaire d'introduire une mesure d_{ij} de similarité entre chaque paire de sommets. Plusieurs choix d'une telle mesure sont possibles ; dans notre cas elle doit s'appuyer sur la structure du graphe.

L'algorithme part d'une structure dans laquelle chaque sommet est identifié à une communauté. On itère alors les étapes suivantes : on calcule des distances entre communautés et on fusionne les deux communautés les plus proches en une nouvelle communauté. Le nombre de communautés diminue de un à chaque étape, le processus s'arrête lorsqu'il n'y a plus qu'une seule communauté correspondant au graphe entier. On obtient ainsi une structure hiérarchique de communautés qui peut être représentée sous une forme arborescente appelée dendrogramme (voir figure 4 p.21) : les feuilles sont les sommets du graphe tandis que les nœuds représentent les communautés créées et sont reliés en fonction des fusions de communautés. La racine de la structure correspond au graphe entier.

Il existe plusieurs façons de calculer la distance entre deux communautés. La plus simple (*single linkage*) considère que la distance entre deux communautés est la distance minimale entre deux sommets de celles-ci. À l'opposé, on peut considérer la distance maximale (*complete linkage*). De manière intermédiaire (*average linkage*) on peut considère que la distance entre deux communautés est la moyenne des distances entre chaque paire de sommets, les deux sommets appartenant chacun à une des deux communautés. Il est encore possible dans certain cas de représenter chaque communauté par un sommet moyen et de considérer leurs distances respectives (*centroïd*). La dernière méthode (dite de *Ward*) [15] que nous utiliserons et expliquerons en 5.3 consiste à minimiser la somme des carrés des distances de chaque sommet au sommet moyen représentant sa communauté.

2.2 Les approches récentes

Le domaine a récemment reçu un vif regain d'intérêt avec l'arrivée de nouveaux algorithmes. Nous allons exposer les principaux algorithmes de détection de communautés

récents. Il y a principalement deux grandes méthodes : l'approche séparative et l'approche agglomérative.

2.2.1 Les approches séparatives

L'idée commune à toutes ces méthodes est d'essayer de scinder le graphe en plusieurs communautés en retirant progressivement les arêtes reliant deux communautés distinctes. Les arêtes sont retirées une à une, à chaque étape les composantes connexes du graphe obtenu sont identifiées à des communautés. Le processus est répété jusqu'au retrait de toutes les arêtes. On obtient alors une structure hiérarchique de communautés. Les méthodes existantes diffèrent par la façon de choisir les arêtes à retirer.

L'algorithme de Girvan et Newman basé sur la centralité d'intermédiarité [16, 9]. Cette approche retire les arêtes de plus forte centralité d'intermédiarité. Cette centralité est définie pour une arête comme le nombre de plus courts chemins passant par cette arête. Il existe en effet peu d'arêtes reliant les différentes communautés et les plus courts chemins entre deux sommets de deux communautés différentes ont de grandes chances de passer par ces arêtes. Un algorithme calculant la centralité de toutes les arêtes en $\mathcal{O}(mn)$ est proposé. Ce calcul est effectué à chaque étape sur le graphe obtenu après retrait des arêtes. La complexité de l'algorithme est donc $\mathcal{O}(m^2n)$. Une variante considérant des marches aléatoires à la place des plus courts chemins est aussi introduite. Elle donne des résultats légèrement meilleurs mais demande encore plus de calculs.

Il est à noter que le même algorithme de calcul de la centralité d'intermédiarité des arêtes a aussi été introduit en même temps et de manière indépendante par [17].

L'algorithme de Radicchi *et al* basé sur le clustering d'arête [18]. La détection des arêtes intercommunautaires est ici basée sur le fait que de telles arêtes sont dans des zones peu clusterisées. Un coefficient de clustering (d'ordre g) d'arêtes est défini comme étant le nombre de cycles de longueur g passant par l'arête divisé par le nombre total de tels cycles possibles (étant donné les degrés des sommets). Cet algorithme retire donc à chaque étape l'arête de plus faible clustering (d'un ordre donné, 3 ou 4 en pratique). Chaque suppression d'arête ne demande alors qu'une mise à jour locale des coefficients de clustering, ce qui lui permet d'être bien plus rapide que le précédent algorithme. La complexité totale est en $\mathcal{O}(m^2)$.

L'algorithme de Fortunato *et al* basé sur la centralité d'information [8]. Il s'agit d'une variante de l'algorithme de Girvan et Newman basé sur une autre notion de centralité : la centralité d'information. Les auteurs définissent l'efficacité de communication ϵ_{ij} entre deux sommets i et j du graphe comme étant l'inverse leur distance (en terme de plus courts chemins). L'efficacité E du réseau est alors définie comme la moyenne des ϵ_{ij} pour tous les couples de sommets. Enfin, la centralité d'information d'une arête est définie comme étant la diminution relative de l'efficacité du réseau lorsque l'on retire cette arête du graphe. Cette approche donne de meilleurs résultats mais a une très mauvaise complexité en $\mathcal{O}(m^3n)$.

2.2.2 Les approches agglomératives

L'idée commune de toutes ces méthodes est d'utiliser une approche s'apparentant à celle du clustering hiérarchique dans lesquelles les sommets sont regroupés itérativement en com-

munautés.

L'algorithme de Newman d'optimisation de la modularité [19]. Newman introduit une notion de modularité ; il s'agit d'une valeur Q quantifiant la qualité d'une partition du graphe en communautés. L'algorithme fusionne alors à chaque étape les communautés permettant d'avoir la plus grande augmentation de la modularité.

Etant donnée une partition $\mathcal{C}$ du graphe en communautés, on définit la fraction e_{ij} d'arêtes du graphe joignant la communauté i à la communauté j (une arête reliant deux communautés distinctes est comptabilisée pour moitié dans e_{ij} et pour moitié dans e_{ji}). Notons $a_i = \sum_j e_{ij}$ la fraction d'arêtes ayant une extrémité dans la communauté i. La modularité Q est alors définie par :

$$Q = \sum_i (e_{ii} - a_i^2) \tag{1}$$

Cette valeur représente la proportion d'arêtes qui sont à l'intérieur d'une communauté moins cette même proportion dans le cas où les arêtes auraient été placées au hasard entre les communautés (en respectant les fractions d'arêtes a_i intervenant dans chaque la communauté i). Cette quantité est retirée pour éviter de compter des arêtes qui ne sont là que pour des raisons statistiques.

Pour améliorer les performances de l'algorithme, seules les communautés ayant une arête entre elles peuvent être fusionnées à chaque étape. Chaque fusion se fait en $\mathcal{O}(n)$ et la mise à jour des valeurs des variations de Q (pour chaque nouvelle fusion possible) peut être effectuée facilement en $\mathcal{O}(m)$. La complexité globale est alors $\mathcal{O}(mn)$. Cette méthode est rapide mais donne de moins bons résultats que les autres.

L'algorithme de Donetti et Muñoz basé sur les propriétés spectrales de la matrice Laplacienne du graphe [20]. La matrice Laplacienne du graphe est (selon les notations qui seront introduites en 3.1) $L = D - A$. Les auteurs observent que les vecteurs propres de cette matrice donnent de l'information sur les communautés. En effet les coordonnés i et j des vecteurs propres correspondant aux plus petites valeurs propres sont corrélées lorsque les sommets i et j sont dans la même communauté. Une distance entre sommets est alors calculée à partir de ces vecteurs propres. Le nombre de vecteurs propres à considérer est a priori inconnu, plusieurs calculs sont successivement effectués en prenant en compte différents nombres de vecteurs propres, et le meilleur résultat est retenu. Les performances de l'algorithme sont limitées par les calcul des vecteurs propres qui se fait en $\mathcal{O}(n^3)$ pour une matrice creuse.

2.3 Notre approche

La méthode que nous proposons peut être classée parmi les approches agglomératives. Nous nous appuyons sur la constatation intuitive qu'une marche aléatoire de courte longueur a tendance à rester piégée dans les communautés. Nous introduisons alors une distance entre deux sommets en nous basant sur le comportement des marches aléatoire partant de ces sommets. Notre distance peut être reliée aux approches spectrales existantes mais ne nécessite pas de calcul de valeurs propres. Nous utilisons un algorithme de clustering hiérarchique basé sur la méthode de Ward. La complexité de notre algorithme est dans les cas favorables en

$\mathcal{O}(mn \ln n)$. Nous proposons aussi une méthode de calcul approché de notre distance permettant d'obtenir des performances meilleures en contrepartie d'une légère perte de précision.

3 Généralités, définitions et notations

3.1 Graphes

Nous considérons un graphe non orienté $G = (V, E)$ possédant $n = |V|$ sommets et $m = |E|$ arêtes. Nous allons considérer de plus que chaque sommet est lié à lui même en ajoutant si nécessaire des boucles sur chaque sommet. Dans le contexte de recherche de communautés, nous supposerons toujours que G est connexe. En effet, deux sommets appartenant à deux composantes connexes différentes ne feront jamais partie d'une même communauté. Un calcul simple (en temps linéaire $\mathcal{O}(n+m)$) des composantes connexes du graphe peut être préalablement effectué et chaque composante connexe obtenue sera alors traitée séparément.

Le graphe peut être représenté par sa matrice d'adjacence A : $A_{ij} = 1$ si les sommets i et j sont reliés par une arête ($\{i, j\} \in E$) et $A_{ij} = 0$ dans le cas contraire. Les sommets étant liés à eux même nous avons $\forall i, A_{ii} = 1$. Le degré $d(i) = \sum_j A_{ij}$ du sommet i est le nombre de ses voisins directs. Pour simplifier l'exposé nous considérerons ici des graphes non pondérés où toutes les arêtes jouent des rôles identiques. Il est cependant immédiat de généraliser notre étude à des graphes pondérés. Pour cela il suffit de prendre en compte le poids des arêtes dans la matrice d'adjacence ($A_{ij} \in \mathbb{R}^+$ au lieu de $A_{ij} \in \{0, 1\}$) et d'adapter la terminologie utilisée.

3.2 Marches aléatoires dans un graphe

Nous allons nous intéresser à un processus de marche aléatoire (ou de diffusion) dans le graphe G. Le temps est discrétisé ($t = 0, 1, 2, \ldots$). À chaque instant, un marcheur est localisé sur un sommet et se déplace à l'instant suivant vers un sommet choisi aléatoirement et uniformément parmi les sommets voisins. La suite des sommets visités est alors une marche aléatoire, et la probabilité de transition du sommet i au sommet j est à chaque étape :

$$P_{ij} = \frac{A_{ij}}{d(i)} \tag{2}$$

Ceci définit la matrice de transition P de la chaîne de Markov correspondant à cette marche aléatoire. Nous pouvons aussi écrire $P = D^{-1}A$ en introduisant la matrice diagonale des degrés des sommets D ($D_{ii} = d(i)$ et $D_{ij} = 0$ pour $i \neq j$).

3.2.1 Dynamique du processus

Etant donnée une position initiale du marcheur, nous définissons la distribution de probabilité de position ρ_t du marcheur après t étapes : il s'agit d'un vecteur dont chacune des n coordonnées $\rho_t(j)$ correspond à la probabilité d'observer le marcheur sur le sommet j à l'instant t. Nous avons donc $\forall j, \rho_t(j) \geq 0$ et $\sum_j \rho_t(j) = 1$. La distribution ρ_0 représente la position initiale du marcheur. Par exemple, s'il part d'un sommet fixé i, $\rho_0(i) = 1$ et $\rho_0(j) = 0$ pour $j \neq i$.

Etant donnée une probabilité de position ρ_t du marcheur à l'instant t, la probabilité de le trouver sur le sommet j à l'instant suivant est :

$$\rho_{t+1}(j) = \sum_i \rho_t(i) \frac{A_{ij}}{d(i)} \tag{3}$$

En utilisant la matrice de transition P définie par l'équation (2), nous pouvons écrire cette équation sous la forme matricielle suivante :

$$\rho_{t+1} = P^T \rho_t \tag{4}$$

Où P^T désigne la transposée de la matrice P. Nous obtenons alors :

$$\rho_t = (P^T)^t \rho_0 \tag{5}$$

Nous noterons $P^t_{ij} = (P^t)_{ij}$ la probabilité d'aller du sommet i au sommet j en t étapes (voir figure 1). Cette probabilité est égale à $\rho_t(j)$ lorsque la position initiale du marcheur est concentrée sur le sommet i ($\rho_0(i) = 1$ et $\rho_0(k) = 0$ pour $k \neq i$).

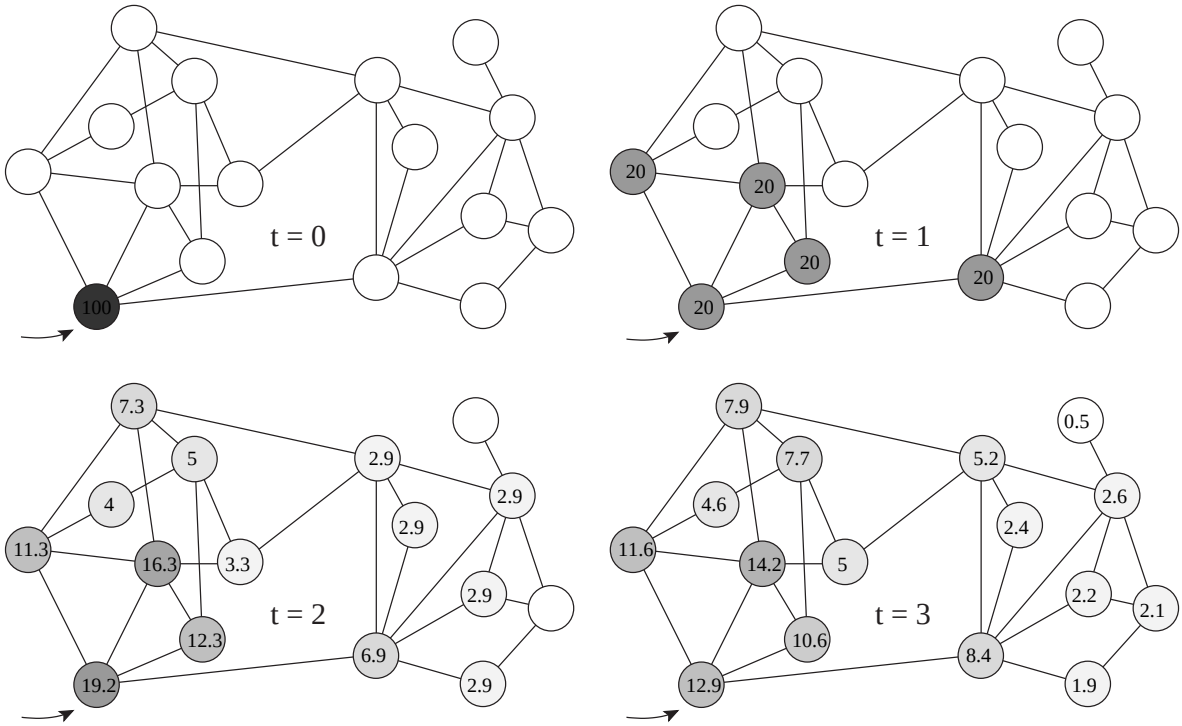


FIG. 1 – Exemple de dynamique de marche aléatoire : le marcheur part du sommet i fléché. Pour chaque temps t la valeur inscrite sur chaque sommet j est la probabilité P^t_{ij} (exprimée en pourcentage) d'observer le marcheur sur ce sommet après t pas.

3.2.2 Propriétés générales

Le graphe G étant non orienté connexe et apériodique (par la présence des boucles sur chaque sommet), la chaîne de Markov sous-jacente est ergodique. Le théorème ergodique

implique l'existence d'une unique probabilité stationnaire de position vérifiant $P^T\pi = \pi$. Il est facile de vérifier que $\pi(i) = \frac{d(i)}{\sum_j d(j)}$. De plus toute marche aléatoire converge (lorsque sa longueur t tend vers l'infini) indépendamment de sa situation initiale ρ_0 vers cette probabilité :

$$\lim_{t\to+\infty} \rho_t = \pi \text{ soit } \lim_{t\to+\infty} \rho_t(i) = \frac{d(i)}{\sum_j d(j)} \tag{6}$$

En d'autres termes, lors d'une marche aléatoire suffisamment longue dans un graphe, la probabilité de se trouver sur un sommet donné est directement (et uniquement) proportionnelle au degré de ce sommet (nous verrons une démonstration de cette propriété en 4.2).

Une autre propriété générale[1] des marches aléatoires dans les graphes non orientés est :

$$d(i)P^t_{ij} = d(j)P^t_{ji} \tag{7}$$

Ceci signifie que la probabilité d'aller de i à j et celle d'aller de j à i par une marche aléatoire de longueur fixée ont un rapport de proportionnalité qui ne dépend que des degrés des sommets de départ et d'arrivée. Ce résultat peut se montrer par récurrence ou en observant que l'égalité peut s'écrire sous la forme matricielle : $DP^tD^{-1} = (P^t)^T$. En utilisant $P = D^{-1}A$ et la symétrie des matrices D et A nous avons alors : $DP^tD^{-1} = D(D^{-1}A)^tD^{-1} = (AD^{-1})^t = (A^T(D^{-1})^T)^t = ((D^{-1}A)^T)^t = (P^t)^T$.

3.3 Quelques rappels d'algèbre linéaire

Nous rappelons brièvement ici les principaux résultats d'algèbre linéaires que nous allons utiliser.

Considérons une matrice $P \in \mathbb{R}^n$, cette matrice possède n *valeurs propres* $\lambda_1 \dots \lambda_n$. Les *vecteurs propres* v_α à droite associés vérifiant $\forall\alpha, Pv_\alpha = \lambda_\alpha v_\alpha$. Les vecteurs propres u_α à gauche vérifient : $P^T u_\alpha = \lambda_\alpha u_\alpha$ où P^T est la transposée de la matrice P.

Une matrice P est dite *semblable* à une matrice Q s'il existe une matrice inversible R telle que $P = R^{-1}QR$. Deux matrices semblables ont les mêmes valeurs propres, en effet si $Pv_\alpha = \lambda_\alpha v_\alpha$ alors $Q(Rv_\alpha) = \lambda_\alpha(Rv_\alpha)$. Ceci nous donne aussi la relation entre les vecteurs propres de P et les vecteurs propres de Q.

Une matrice S est dite symétrique si $S^T = S$. Une matrice réelle *symétrique* est diagonalisable dans une base orthonormale de vecteurs propres. C'est à dire qu'il existe une famille s_α de vecteurs propres de S vérifiant : $\forall\alpha, Ss_\alpha = \lambda_\alpha s_\alpha$ et $\forall\alpha, \forall\beta, s^T_\alpha s_\beta = \delta_{\alpha\beta}$ où $\delta_{\alpha\beta}$ vaut 1 si $\alpha = \beta$ et 0 sinon. Nous pouvons effectuer une décomposition spectrale d'une telle matrice pour obtenir l'expression suivante :

$$S = \sum_{\alpha=1}^{n} \lambda_\alpha s_\alpha s^T_\alpha \tag{8}$$

Cette décomposition permet de calculer simplement les puissances de la matrice S :

$$S^t = \sum_{\alpha=1}^{n} \lambda^t_\alpha s_\alpha s^T_\alpha \tag{9}$$

Une matrice $P \in \mathbb{R}^n$ est dite stochastique si chaque élément de la matrice est positif et que la somme des éléments d'une colonne vaut 1 : $\forall i, \forall j, 0 \leq P_{ij} \leq 1$ et $\sum_{j=1}^n P_{ij} = 1$. Les

[1] Cette propriété correspond en fait à la propriété de réversibilité de la chaîne de Markov sous-jacente.

valeurs propres d'une telle matrice sont inférieure à 1 en module : $\forall \alpha, |\lambda_\alpha| \leq 1$. La valeur propre principale est toujours $\lambda_1 = 1$ et elle est associée à un vecteur propre à droite constant $\forall i, \forall j, v_1(i) = v_1(j)$.

L'exponentielle d'une matrice M est la matrice définie par la série :

$$e^M = \sum_{k=0}^{\infty} \frac{M^k}{k!} \tag{10}$$

Les exponentielles de matrices interviennent naturellement dans la résolution de systèmes linéaires d'équations différentielles. En effet, l'équation $y'(t) = My(t)$ (où $y(t)$ est un vecteur de dimension n) a pour solution :

$$y(t) = e^{tM} y(0) \tag{11}$$

4 Evaluation de la similarité et de la différence des sommets grâce à des marches aléatoires courtes

Afin de pouvoir grouper les sommets du graphe par communautés, nous allons introduire une distance entre sommets qui évalue leur proximité, dans le sens où la distance entre deux sommets d'une même communauté doit être beaucoup plus faible que la distance entre deux sommets appartenant à deux communautés distinctes.

4.1 Communautés et marches aléatoires courtes

Nous avons vu que les marches aléatoires suffisamment longues "oublient" leur position initiale car la distribution limite de probabilité de position est indépendante du point de départ de la marche aléatoire. Par contre, une marche aléatoire plus courte a tendance à rester piégée dans les zones plus denses correspondant aux communautés. C'est en se basant sur cette constatation intuitive (déjà soulignée dans [21]) que nous allons capturer des informations sur la similarité des sommets.

Nous allons donc considérer des marches aléatoires de longueur fixée t suffisamment courte pour ne pas atteindre le régime stationnaire mais suffisamment longue pour collecter des informations globales sur le voisinage du point de départ de la marche (le choix de la longueur t des marches aléatoires est délicat et sera discuté en 4.6). Nous supposons donc connaître pour tous sommets i et j les probabilités P^t_{ij} d'aller de i à j en t pas (le calcul effectif de ces probabilités sera exposé en 4.5).

Chaque probabilité P^t_{ij} apporte de l'information sur i et j. Ainsi si nous voulons comparer deux sommets i et j nous utiliserons l'information fournie par les probabilités P^t_{ik}, P^t_{jk}, P^t_{ki} et P^t_{kj} pour tout sommet k. Cependant, l'équation (7) reliant les probabilités P^t_{ik} et P^t_{ki} nous montre que ces deux probabilités apportent exactement la même information. Pour comparer les sommets i et j nous pourrons donc nous contenter d'examiner les probabilités P^t_{ik} et P^t_{jk}. Ceci revient à comparer les vecteurs $P^t_{i\bullet}$ et $P^t_{j\bullet}$ correspondant aux lignes i et j de la matrice P^t. Finalement, chaque sommet i sera de notre point de vue pleinement caractérisé par le vecteur $P^t_{i\bullet}$.

La façon de comparer ces données doit s'appuyer sur les constatations suivantes :

– Si deux sommets i et j sont dans une même communauté, la probabilité P^t_{ij} sera très certainement élevée. Réciproquement, si P^t_{ij} est élevée il n'est pas toujours garanti que i et j soient dans la même communauté.

– La probabilité P_{ij}^t est influencée par le degré $d(j)$ du sommet d'arrivée : les marches aléatoires ont plus de chances de passer par les sommets de fort degré (dans le cas limite d'une marche aléatoire infinie, cette probabilité est proportionnelle au degré).
– Les sommets d'une même communauté ont tendance à voir les sommets éloignés de la même façon, ainsi si i et j sont dans la même communauté et k dans une autre communauté il y a de fortes chances que $P_{ik}^t \simeq P_{jk}^t$.

4.2 Etude spectrale de la matrice de transfert

L'équation (5) montre que les marches aléatoires dans un graphe sont régies par les puissances de la transposée de la matrice de transition P. Pour mieux comprendre les mécanismes mis en jeu, nous sommes donc poussés à effectuer une analyse spectrale de la matrice P. Pour une étude plus complète, nous renvoyons à [22].

Tout d'abord, remarquons que les valeurs propres de la matrice P sont réelles. En effet, P est semblable à la matrice symétrique :

$$S = D^{\frac{1}{2}} P D^{-\frac{1}{2}} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{12}$$

Nous classerons les valeurs propres de P (qui sont les mêmes que celles de S) par ordre décroissant de module $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$. La matrice P est une matrice stochastique ; sa valeur propre principale est donc $\lambda_1 = 1$. Comme le graphe G est connexe et acyclique, le théorème de Perron-Frobenius nous dit que λ_1 est la seule valeur propre principale : $|\lambda_2| < 1$. Les valeurs propres de P vérifient donc :

$$1 = \lambda_1 > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n| \tag{13}$$

La matrice S étant symétrique, nous pouvons considérer une famille $(s_\alpha)_{1 \leq \alpha \leq n}$ de vecteurs propres orthonormés de S. Nous avons ainsi pour tous α et β : $S s_\alpha = \lambda_\alpha s_\alpha$ et $s_\alpha^T s_\beta = \delta_{\alpha\beta}$ (où $\delta_{\alpha\beta}$ vaut 1 si $\alpha = \beta$ et 0 sinon).

Les vecteurs propres de P sont reliés à ceux de S : les vecteurs $v_\alpha = D^{-\frac{1}{2}} s_\alpha$ constituent une famille de vecteurs propres à droite de P et les vecteurs $u_\alpha = D^{\frac{1}{2}} s_\alpha$ constituent une famille de vecteurs propres à gauche de P. Nous avons alors pour tous α et β :

$$P v_\alpha = \lambda_\alpha v_\alpha \text{ et } P^T u_\alpha = \lambda_\alpha u_\alpha \tag{14}$$

$$u_\alpha^T v_\beta = \delta_{\alpha\beta} \tag{15}$$

Nous obtenons ainsi une décomposition spectrale de la matrice P^T :

$$P^T = \sum_{\alpha=1}^{n} \lambda_\alpha u_\alpha v_\alpha^T = D^{\frac{1}{2}} \sum_{\alpha=1}^{n} \lambda_\alpha s_\alpha v_\alpha^T \tag{16}$$

Et donc :

$$(P^T)^t = \sum_{\alpha=1}^{n} \lambda_\alpha^t u_\alpha v_\alpha^T = D^{\frac{1}{2}} \sum_{\alpha=1}^{n} \lambda_\alpha^t s_\alpha v_\alpha^T \tag{17}$$

Le vecteur de probabilité de position ρ_t défini par l'équation (5) vérifie donc :

$$\rho_t = (P^T)^t \rho_0 = \sum_{\alpha=1}^{n} \lambda_\alpha^t u_\alpha (v_\alpha^T \rho_0) = D^{\frac{1}{2}} \sum_{\alpha=1}^{n} \lambda_\alpha^t s_\alpha (v_\alpha^T \rho_0) \tag{18}$$

Ceci montre par exemple que lorsque t tend vers l'infini, le comportement des marches aléatoires n'est dicté que par la valeur propre λ_1 et par son vecteur propre associé. Il est facile de vérifier que v_1 est un vecteur constant et nous obtenons en normalisant $v_1(i) = \frac{1}{\sqrt{\sum_j d(j)}}$ et $u_1(i) = \frac{d(i)}{\sqrt{\sum_j d(j)}}$ pour tout sommet i. Nous pouvons alors retrouver la propriété de distribution limite des marches aléatoires, équation (6) :

$$\lim_{t\to+\infty} \rho_t(i) = \lim_{t\to+\infty} \sum_{\alpha=1}^{n} \lambda_\alpha^t (v_\alpha^T \rho_0) u_\alpha(i) = (v_1^T \rho_0) u_1(i) = \frac{d(i)}{\sum_j d(j)} \tag{19}$$

Nous obtenons aussi l'expression analytique de la probabilité P_{ij}^t d'aller du sommet i au sommet j par une marche aléatoire de longueur t :

$$P_{ij}^t = \sum_{\alpha=1}^{n} \lambda_\alpha^t u_\alpha(j) v_\alpha(i) = \sqrt{d(j)} \sum_{\alpha=1}^{n} \lambda_\alpha^t s_\alpha(j) v_\alpha(i) \tag{20}$$

Ceci donne pour les lignes $P_{i\bullet}^t$ de la matrice P^t :

$$P_{i\bullet}^t = \sum_{\alpha=1}^{n} \lambda_\alpha^t v_\alpha(i) u_\alpha = D^{\frac{1}{2}} \sum_{\alpha=1}^{n} \lambda_\alpha^t v_\alpha(i) s_\alpha \tag{21}$$

4.3 Définition de la distance entre sommets

Les considérations de 4.1 nous ont montré que l'évaluation de la proximité de deux sommets i et j peut se faire en comparant les vecteurs $P_{i\bullet}^t$ et $P_{j\bullet}^t$ (la figure 2 compare sur un exemple ces vecteurs pour trois sommets de départ différents). Nous avons aussi noté qu'il faut tenir compte de l'influence du degré $d(k)$ du sommet d'arrivée sur la probabilité P_{ik}^t. Aux vues de ces considérations et de l'analyse faite en 4.2, nous sommes incités à considérer la distance r_{ij} quantifiant la proximité des sommets i et j définie par :

$$r_{ij} = \left\| D^{-\frac{1}{2}} P_{i\bullet}^t - D^{-\frac{1}{2}} P_{j\bullet}^t \right\| = \sqrt{\sum_{k=1}^{n} \frac{(P_{ik}^t - P_{jk}^t)^2}{d(k)}} \tag{22}$$

où $\|.\|$ est la norme euclidienne canonique (notons au passage que r est une norme euclidienne sur l'espace $\mathbb{R}^n$ contenant les vecteurs $P_{i\bullet}^t$). L'équation (21) nous permet de reformuler notre distance :

$$r_{ij} = \left\| \sum_{\alpha=1}^{n} \lambda_\alpha^t (v_\alpha(i) - v_\alpha(j)) s_\alpha \right\| \tag{23}$$

Le vecteur v_1 étant constant ($\forall i, j, v_1(i) = v_1(j)$), la somme peut ignorer le cas $\alpha = 1$. De plus la famille de vecteurs s_α étant orthonormée, le théorème de Pythagore nous permet de simplifier l'expression de notre distance par :

$$r_{ij}^2 = \sum_{\alpha=2}^{n} \lambda_\alpha^{2t} (v_\alpha(i) - v_\alpha(j))^2 \tag{24}$$

Cette distance compare donc les coordonnées des vecteurs propres à droite v_α de P et pondère celles-ci par λ_α^t. Ainsi la somme est dominée par les termes liés aux plus grandes valeurs propres.

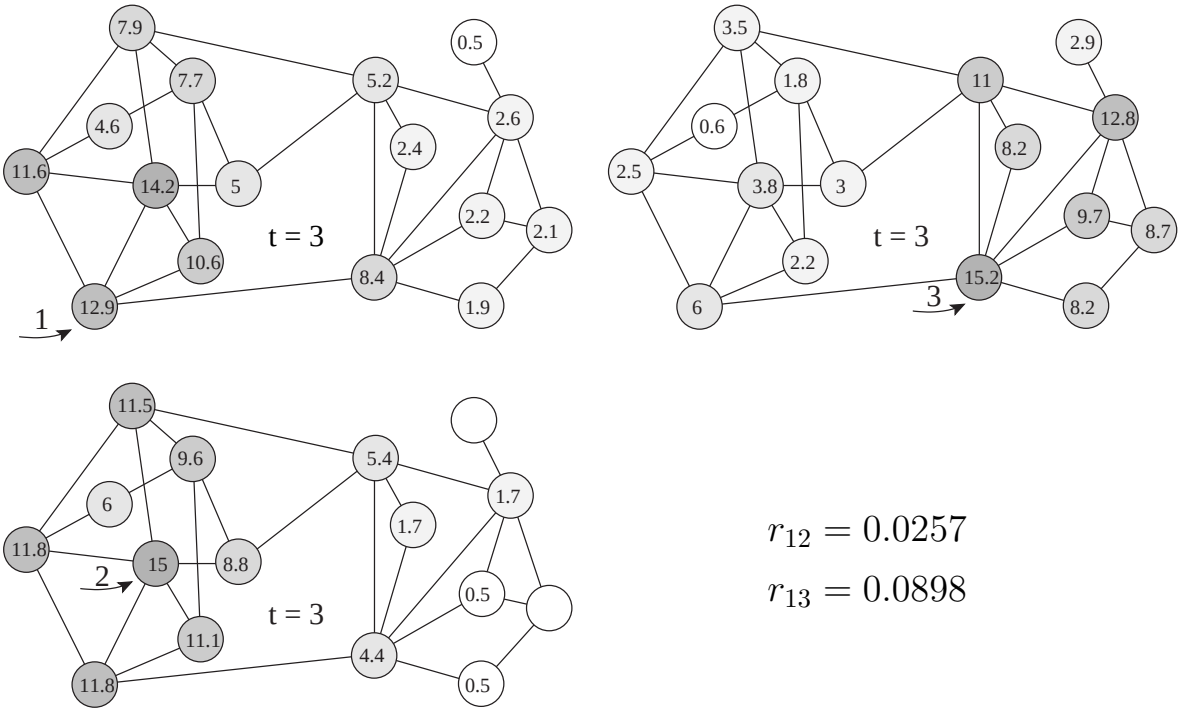


FIG. 2 – Exemple de calcul de la distance r. Nous montrons des marches aléatoires de longueur $t = 3$ partant de trois sommets différents : le sommet de départ est fléché et les probabilités P_{ij}^t nécessaires au calcul de la distance sont indiquées sur chaque sommet d'arrivée.

Soulignons que de nombreux travaux ont montré ou constaté que les propriétés structurelles du graphe sont capturées dans les vecteurs propres correspondant aux plus grandes valeurs propres. Par exemple, [23] observe que les vecteurs propres (autres que v_1) correspondant aux plus grandes valeurs propres positives traduisent la structure modulaire du graphe au sens où deux sommets d'une même communauté ont des coordonnées similaires dans chacun de ces vecteurs propres. De même [24, 25] montrent dans un cadre plus général que lorsque les valeurs propres tendent vers 1, les vecteurs propres associés sont constants par morceaux, chaque morceau correspondant à une communauté. Une distance similaire à la notre y est aussi introduite. Enfin, [20] utilise cette même approche spectrale appliquée à la matrice Laplacienne du graphe ($L = D - A$).

Notons par ailleurs que [23] met en évidence que les valeurs propres négatives de module important correspondent à des phénomènes oscillatoires de diffusion et non à des propriétés structurelles du graphe. Il serait donc souhaitable qu'elles jouent un rôle minime dans le calcul de notre distance. Nous allons dans cette optique proposer une solution à ce problème en 4.6.

Toutes ces études montrent que l'approche spectrale peut jouer un rôle important dans l'étude de la structure communautaire d'un graphe. Cependant ces approches ont toutes l'inconvénient de devoir calculer explicitement les valeurs propres et les vecteurs propres associés. Ce calcul devient rapidement impraticable lorsque la taille du graphe dépasse quelques milliers de sommets. Notre approche s'appuie sur les mêmes fondements théoriques mais vise à s'affranchir du calcul pénalisant des valeurs propres, comme nous allons le voir en 4.5.

4.4 Distance entre communautés

Nous pouvons naturellement généraliser notre distance entre sommets en une distance entre communautés. Pour toute communauté $C \subset V$. Nous pouvons définir une marche aléatoire partant uniformément de C en choisissant comme distribution initiale $\rho_0(i) = \frac{1}{|C|}$ si $i \in C$ et $\rho_0(i) = 0$ sinon. Pour tout sommet j, la probabilité P^t_{Cj} d'atteindre j en partant de la communauté C en t étapes est alors définie par :

$$P^t_{Cj} = \rho_t(j) = \frac{1}{|C|} \sum_{i \in C} P^t_{ij} \tag{25}$$

Nous obtenons alors un vecteur $P^t_{C\bullet}$ caractérisant la communauté C. Nous introduisons la distance $r_{C_1C_2}$ entre deux communautés C_1 et C_2 :

$$r_{C_1C_2} = \left\| D^{-\frac{1}{2}} P^t_{C_1\bullet} - D^{-\frac{1}{2}} P^t_{C_2\bullet} \right\| = \sqrt{\sum_{k=1}^{n} \frac{(P^t_{C_1k} - P^t_{C_2k})^2}{d(k)}} \tag{26}$$

Cette définition est bien une généralisation de la distance entre sommets : elles coïncident si l'on assimile un sommet à la communauté réduite à ce sommet.

4.5 Calcul de la distance

La définition donné par l'équation (22) de r_{ij} permet un calcul efficace de celle-ci à partir des probabilités P_{ik} et P_{jk}. En effet il s'agit principalement d'une somme de n termes faisant intervenir ces probabilités et les degrés des sommets. Le calcul des probabilités peut alors être fait de manière exacte ou approchée.

4.5.1 Calcul exact

Le calcul exact des différentes probabilités P^t_{ik} peut être fait en calculant la matrice P^t. Les graphes rencontrés en pratique sont généralement peu denses, il est alors possible de calculer chaque vecteur $P^t_{i\bullet}$ de manière efficace en multipliant t fois le vecteur $P^0_{i\bullet}$ ($P^0_{i\bullet}(k) = P^0_{ik} = \delta_{ik}$) par P^T. Chaque multiplication demande $\mathcal{O}(m)$ et l'initialisation $\mathcal{O}(n)$. Le calcul de $P^t_{i\bullet}$ demande donc un temps $\mathcal{O}(n + tm)$ qui peut s'écrire $\mathcal{O}(tm)$ car $m \geq n$ (le graphe G est connexe).

Chaque calcul de distance r_{ij} effectué selon l'équation (22) demande alors un temps supplémentaire en $\mathcal{O}(n)$.

4.5.2 Calcul approché

Le calcul exact peut s'avérer trop coûteux pour les très grands graphes. Il peut alors être pertinent de calculer les probabilités P_{ik} de manière approchée. Pour estimer cette probabilité il suffit de simuler un certain nombre K de marches aléatoires de longueur t partant du sommet i. Chaque simulation est réalisée en un temps $\mathcal{O}(t)$. Nous comptabilisons alors le nombre N_{ik} de marches aléatoires ayant fini au sommet k. Nous estimons finalement les probabilités P^t_{ik} d'aller de i à k en t étapes par :

$$\widetilde{P}^t_{ik} = \frac{N_{ik}}{K} \tag{27}$$

Le théorème central limite indique alors que $\widetilde{P}^t_{ik}$ tend vers P^t_{ik} avec une erreur relative qui décroît à une vitesse $\mathcal{O}(\frac{1}{\sqrt{K}})$ lorsque K tend vers l'infini. Le choix de K dépend alors de la précision voulue et du temps de calcul disponible. Cette approche n'est avantageuse que pour de très grands graphes lorsque $K < n$.

Lorsque nous effectuons les simulations, nous devons actualiser les compteurs N_{ik}. Si $K < n$, nous allouons dynamiquement de la mémoire uniquement pour les compteurs effectivement utilisés. La structure de données utilisée pour la gestion des ces compteurs sera un arbre équilibré. La taille de l'arbre étant majorée par K, chaque mise à jour et chaque ajout d'un nouveau compteur se fait en temps $\mathcal{O}(\ln K)$. Le temps total du calcul de $\widetilde{P}^t_{i\bullet}$ est donc $\mathcal{O}(K(t+\ln K))$ en utilisant un espace $\mathcal{O}(K)$. Une fois les simulations effectuées, nous ne stockons en mémoire que les valeurs $\widetilde{P}^t_{ik}$ non nulles, ce qui permet ainsi d'utiliser un espace mémoire $\mathcal{O}(K)$ seulement pour chaque vecteur $\widetilde{P}^t_{i\bullet}$.

Les vecteurs estimés $\widetilde{P}^t_{i\bullet}$ et $\widetilde{P}^t_{j\bullet}$ n'ayant qu'au plus K composantes non nulles chacun, chaque calcul de r_{ij} demande alors un temps supplémentaire $\mathcal{O}(K)$. Lorsque K et t sont fixés, nous obtenons une complexité constante. Mais la constante est grande car en pratique K doit au moins être de l'ordre de 1000 pour obtenir une précision suffisante. L'influence du choix de K sera illustrée en 6.4.

4.5.3 Gestion de la mémoire

Pour éviter de faire plusieurs fois les mêmes calculs, nous pouvons stocker en mémoire la matrice P^t. La matrice peut être calculée de manière exacte en un temps total $\mathcal{O}(tmn)$ pour un espace mémoire utilisé en $\mathcal{O}(n^2)$. Le calcul approché nécessite lui un temps total $\mathcal{O}(nK(t+\ln K))$ et un espace mémoire $\mathcal{O}(nK)$. Le calcul de chaque distance r_{ij} nécessite respectivement $\mathcal{O}(n)$ et $\mathcal{O}(K)$.

Cependant pour de très grands graphes il est impossible de stocker la matrice P^t entière. Il reste possible de calculer chaque distance r_{ij} en un temps $\mathcal{O}(tm)$ de manière exacte ou $\mathcal{O}(K(t+\ln K))$ de manière approchée sans la stocker.

Une approche intermédiaire stockant partiellement la matrice P^t peut aussi être envisagée. Dans tous les cas, il est judicieux que l'algorithme de clustering utilisé calcule ces distances uniquement lorsqu'elles sont effectivement requises.

4.6 Choix de la longueur de la marche aléatoire

La longueur t des marches aléatoires considérées est un paramètre essentiel de l'algorithme. En effet si t est trop grand, les probabilités P^t_{ik} vont tendre vers la même probabilité limite et il sera alors impossible d'apprécier les différences entre deux sommets car r_{ij} tendra vers 0 pour tout couple de sommets i et j. Au contraire si t est trop petit, les marches aléatoires ne fournissent pas assez d'information sur le voisinage des sommets. Le cas extrême où $t = 0$ conduit à $r_{ij} = 1$ pour tout couple de sommets i et j. Ceci correspond à une incapacité d'apprécier les similarités entre deux sommets.

En résumé le choix de t dépend de la taille des communautés à détecter. Il doit être un compromis entre le choix $t = 0$ qui conduit à n minuscules communautés (d'un seul sommet) et $t = +\infty$ qui conduit à une seule communauté (regroupant tous les sommets). L'influence de t en pratique sera illustrée dans 6.3.

Un autre problème peut se poser : le temps est discrétisé et nous sommes donc obligés de choisir $t \in \mathbb{N}$. Ceci peut représenter trop peu de liberté car comme nous le verrons en 6.3, t

est en pratique petit (de l'ordre de quelques unités). Il est possible par exemple que $t = 3$ soit trop petit et $t = 4$ soit déjà trop grand. Nous aimerions pouvoir choisir des t intermédiaires. La solution consiste à transformer notre processus de marche aléatoire qui est une chaîne de Markov en temps discret en une chaîne de Markov en temps continu.

4.6.1 Le processus en temps continu

Considérons une distribution de probabilités de position en temps continu $(\bar{\rho}_t)_{t\in\mathbb{R}^+}$. Pendant une durée infinitésimale dt, le marcheur va changer de sommet selon la matrice de transition P avec une probabilité dt et va rester sur le même sommet avec une probabilité $1 - dt$. La variation de probabilité de position sur chaque sommet j doit prendre en compte la probabilité que le marcheur arrive d'un sommet voisin et celle que le marcheur quitte le sommet durant dt. Ainsi :

$$\frac{d\bar{\rho}_t(j)}{dt} = \sum_{i=1}^{n} \bar{\rho}_t(i)P_{ij} - \bar{\rho}_t(j) \tag{28}$$

Ce qui donne sous forme matricielle :

$$\frac{d\bar{\rho}_t}{dt} = P^T\bar{\rho}_t - \bar{\rho}_t = (P^T - Id)\bar{\rho}_t \tag{29}$$

Cette équation différentielle linéaire du premier ordre se résout facilement et nous donne la chaîne de Markov en temps continu associée :

$$\bar{\rho}_t = e^{t(P^T-Id)}\rho_0 \tag{30}$$

Ce processus a les mêmes propriétés que le processus initial car les vecteurs propres de la matrice $e^{t(P^T-Id)}$ sont les mêmes que ceux de la matrice P^T et les valeurs propres $\bar{\lambda}_\alpha^{(t)}$ sont déduites de celles de la matrice P^T par : $\bar{\lambda}_\alpha^{(t)} = e^{t(\lambda_\alpha-1)}$. Nous pouvons donc écrire une décomposition spectrale de $e^{t(P^T-Id)}$:

$$e^{t(P^T-Id)} = \sum_{\alpha=1}^{n} e^{t(\lambda_\alpha-1)}u_\alpha v_\alpha^T = D^{\frac{1}{2}}\sum_{\alpha=1}^{n} e^{t(\lambda_\alpha-1)}s_\alpha v_\alpha^T \tag{31}$$

Nous voyons que seules les valeurs propres changent par rapport à la décomposition spectrale de la matrice $(P^T)^t$ donnée par l'équation (17). La valeur propre principale est toujours $e^{t(\lambda_1-1)} = e^0 = 1$ comme précédemment, ce qui conduit à la même distribution stationnaire limite. Les quantités $e^{\lambda_\alpha-1}$ conservent le même ordre que les valeur propres λ_α mais sont toutes positives : les valeurs propres négatives λ_α correspondant à des phénomènes oscillatoires indésirables deviennent donc maintenant négligeables. De plus les valeurs propres $e^{t(\lambda_\alpha-1)}$ conservent le même type de décroissance exponentielle que λ_α^t. Ceci justifie l'utilisation du processus en temps continu à la place du processus en temps discret. La distance $\bar{r}_{ij}$ associée sera alors tout simplement définie par :

$$\bar{r}_{ij}^2 = \|D^{-\frac{1}{2}}\bar{P}_{i\bullet}^t - D^{-\frac{1}{2}}\bar{P}_{j\bullet}^t\|^2 = \sum_{\alpha=2}^{n} e^{2t(\lambda_\alpha-1)}(v_\alpha(i) - v_\alpha(j))^2 \tag{32}$$

où $\bar{P}_{i\bullet}^t$ est la $i^{\text{ème}}$ ligne de la matrice $e^{t(P^T-Id)}$.

4.6.2 Calcul des probabilités $\bar{P}_{ij}^t$ pour le processus en temps continu

L'exponentielle de la matrice creuse $t(P^T - Id)$ provenant d'une chaîne de Markov peut être calculée de la manière suivante [26] :

$$e^{t(P^T - Id)} \simeq e^{-t} \sum_{k=0}^{r} \frac{t^k}{k!} (P^T)^k \tag{33}$$

La série est tronquée à un rang r choisi pour obtenir la précision souhaitée : l'erreur commise sur chaque coordonnée est au plus $\varepsilon = e^{-t} \sum_{k=r+1}^{+\infty} \frac{t^k}{k!}$. Cette méthode simple est numériquement stable pour une matrice P stochastique et est efficace pour des graphes peu denses. Le calcul se fait en temps $\mathcal{O}(rnm)$ en calculant successivement les puissances de la matrice P^T. Il est aussi possible de calculer comme précédemment chaque vecteur $\bar{P}_{i\bullet}^t$ séparément en temps $\mathcal{O}(rm)$.

Mode de calcul	Temps de calcul de $P_{i\bullet}^t$	Erreur commise	Temps de calcul de r_{ij}
Exact	$\mathcal{O}(tm)$	0	$\mathcal{O}(n)$
Approché	$\mathcal{O}(K(t + \ln K))$	$\mathcal{O}\Big(\frac{1}{\sqrt{K}}\Big)$	$\mathcal{O}(K)$
Temps continu	$\mathcal{O}(rm)$	$e^{-t} \sum_{k=r+1}^{+\infty} \frac{t^k}{k!}$	$\mathcal{O}(n)$

Tab. 1 – Récapitulatif des complexités des différentes méthodes de calcul

4.6.3 Méthode intermédiaire d'approximation du temps continu

Les calculs imposés par le processus en temps continu sont incompatibles avec les calculs approchés en temps constant proposés en 4.5.2. Il est toutefois possible de considérer une approche intermédiaire entre le temps discret initial et le temps continu. Pour cela nous introduisons un nouveau processus de marche aléatoire en temps discret plus lent : le marcheur à chaque étape a la possibilité de changer de sommet avec une probabilité p ou de rester sur place avec une probabilité $1 - p$, pour un p donné. L'étude que nous avons réalisée reste la même en considérant maintenant la matrice de transition $pP + (1-p)Id$. Les vecteurs propres de cette nouvelle matrice sont les mêmes que ceux de P et sont associés aux valeurs propres $1 - p(1 - \lambda_\alpha)$.

Nous retrouvons donc le processus initial dans les cas où $p = 1$. À l'opposé, le processus en temps continu correspond au cas limite $p \to 0$ avec des marches de longueur $[\frac{t}{p}]$ ($[.]$ désigne la fonction partie entière). En effet :

$$\lim_{p \to 0} (1 - p(1 - \lambda_\alpha))^{[\frac{t}{p}]} = e^{t(\lambda_\alpha - 1)} \tag{34}$$

Nous pouvons donc intuitivement dire que lorsque p et t sont fixés, cette approche donne des résultats équivalents au processus en temps continu avec un temps de l'ordre de grandeur de pt. Nous pouvons finalement obtenir des résultats correspondant à des valeurs de t réelles en contrepartie de calculs plus longs dus à des marches de longueur t plus importantes.

5 Description de l'algorithme

Nous avons maintenant une distance entre sommets ou communautés reflétant la proximité de ceux-ci au niveau structurel du graphe. Le but est donc de construire une structure de communautés en accord avec notre distance. Nous allons pour ceci proposer une variante de la méthode de clustering hiérarchique de Ward (cf 2.1.2).

5.1 Principe de notre algorithme

Dans l'optique de la recherche de communautés dans des graphes de grande taille il est essentiel de proposer un algorithme de clustering peu gourmand en temps de calcul et faisant appel à un minimum de calculs élémentaires de distance. Nous allons proposer ici un tel algorithme basé sur une approche agglomérative.

Nous allons partir d'une partition du graphe en n communautés contenant chacune un seul sommet et faire évoluer cette partition jusqu'à obtenir une seule communauté correspondant au graphe entier. Pour ce faire nous allons répéter les étapes suivantes :

- Choisir deux communautés à fusionner en fonction de leurs distances respectives.
- Fusionner ces deux communautés en une nouvelle communauté.
- Mettre à jour les distances entre communautés.
- Calculer et mémoriser un paramètre quantifiant la qualité de la nouvelle partition.

Nous effectuons ainsi exactement $n - 1$ étapes. Chaque étape nous donne une partition du graphe en communautés et nous choisissons finalement la meilleure partition selon le paramètre de qualité calculé à chaque étape.

5.2 Evaluation de la qualité d'une partition du graphe en communautés

Nous utilisons pour évaluer la qualité d'une partition du graphe en communautés la modularité introduite dans [9, 19]. Il s'agit de la quantité Q déjà définie en 2.2.2 :

$$Q = \sum_i (e_{ii} - a_i^2) \tag{35}$$

où e_{ij} est la fraction d'arêtes du graphe joignant la communauté i à la communauté j et $a_i = \sum_j e_{ij}$ est la fraction d'arêtes ayant une extrémité dans la communauté i.

Cette quantité est calculable en un temps $\mathcal{O}(m)$ et peut être mise à jour après chaque fusion en $\mathcal{O}(1)$. Le coût total du calcul de la modularité lors de l'exécution de l'algorithme est donc $\mathcal{O}(m)$.

Nous retenons comme résultat de notre algorithme la partition du graphe possédant la meilleure modularité.

5.3 Choix des communautés à fusionner

Il faut choisir à chaque étape deux communautés à fusionner. Pour diminuer le nombre de cas à considérer et ainsi la complexité, nous envisagerons uniquement les fusions de communautés ayant au moins une arête entre elles. Cette heuristique (déjà utilisée dans [19] et [20]) permet de ne considérer au plus que m fusions au lieu de n^2 et assure en plus que chaque communauté obtenue est connexe.

Pour choisir les deux communautés à fusionner, nous utilisons le principe de l'algorithme de clustering hiérarchique suivant la méthode de Ward [15]. Cette méthode fusionne les communautés de telle sorte à minimiser la moyenne σ de la distance au carré de chaque sommet à sa communauté :

$$\sigma = \frac{1}{n} \sum_{C} \sum_{i \in C} r_{iC}^2 \tag{36}$$

Ceci évite d'avoir des sommets trop éloignés dans une même communauté et favorise l'apparition de communautés de tailles équilibrées et défavorise l'apparition rapide de trop grandes communautés. Ceci est un avantage sur les autres méthodes de clustering hiérarchiques qui souffrent souvent de phénomènes d'agglomération en chaîne de sommets en une grande communauté. Ces autres méthodes ont ainsi le double désavantage de mal identifier les petites communautés et de favoriser le pire cas en terme de complexité.

L'autre grand avantage de la méthode de Ward est qu'elle s'adapte parfaitement à notre distance r qui est une norme Euclidienne sur l'espace $\mathbb{R}^n$ contenant les vecteurs $P^t_{i\bullet}$ et $P^t_{C\bullet}$. Nous allons montrer que la variation $\Delta\sigma(C_1, C_2)$ de la distance au carré moyenne lors d'une fusion de C_1 et C_2 est très simple à calculer. Lorsque nous fusionnons deux communautés C_1 et C_2 en une nouvelle communauté $C = C_1 \cup C_2$, la variation de σ est :

$$\Delta\sigma(C_1, C_2) = \frac{1}{n}(\sum_{i \in C} r_{iC}^2 - \sum_{i \in C_1} r_{iC_1}^2 - \sum_{i \in C_2} r_{iC_2}^2) \tag{37}$$

Commençons par réorganiser la première somme en utilisant le fait que $C = C_1 \cup C_2$ est une union disjointe :

$$\sum_{i \in C} r_{iC}^2 = \sum_{i \in C_1} r_{iC}^2 + \sum_{i \in C_2} r_{iC}^2 \tag{38}$$

Notons $< .|. >$ le produit scalaire associé à la norme Euclidienne r. Ainsi par exemple $r_{iC}^2 =< P^t_{C\bullet} - P^t_{i\bullet} | P^t_{C\bullet} - P^t_{i\bullet} >$ Pour simplifier les notations, nous allons utiliser des notations vectorielles : posons pour toute communauté C et tout sommet i : $\overrightarrow{iC} = P^t_{C\bullet} - P^t_{i\bullet}$ et de même pour deux communautés $\overrightarrow{C_1C_2} = P^t_{C_2\bullet} - P^t_{C_1\bullet}$. Nous pouvons alors réécrire :

$$\sum_{i \in C_1} r_{iC}^2 = \sum_{i \in C_1} < \overrightarrow{iC} | \overrightarrow{iC} > \tag{39}$$

Faisons intervenir les barycentres $P^t_{C_1\bullet}$ des ensembles de vecteurs $\{P^t_{i\bullet} | i \in C_1\}$:

$$\sum_{i \in C_1} r_{iC}^2 = \sum_{i \in C_1} < \overrightarrow{iC_1} + \overrightarrow{C_1C} | \overrightarrow{iC_1} + \overrightarrow{C_1C} > \tag{40}$$

$$\sum_{i \in C_1} r_{iC}^2 = \sum_{i \in C_1} (< \overrightarrow{iC_1} | \overrightarrow{iC_1} > +2 < \overrightarrow{iC_1} | \overrightarrow{C_1C} > + < \overrightarrow{C_1C} | \overrightarrow{C_1C} >) \tag{41}$$

$$\sum_{i \in C_1} r_{iC}^2 = \sum_{i \in C_1} r_{iC_1}^2 + 2 < \sum_{i \in C_1} \overrightarrow{iC_1} | \overrightarrow{C_1C} > + |C_1| < \overrightarrow{C_1C} | \overrightarrow{C_1C} > \tag{42}$$

comme $P^t_{C_1\bullet}$ est le barycentre de $\{P^t_{i\bullet} | i \in C_1\}$ nous avons :

$$\sum_{i \in C_1} \overrightarrow{iC_1} = \overrightarrow{0} \tag{43}$$

Et de plus, $P^t_{C\bullet}$ est le barycentre de $P_{C_1\bullet}$ pondéré par $|C_1|$ et de $P_{C_2\bullet}$ pondéré par $|C_2|$. Nous avons donc $\overrightarrow{C_1C} = \frac{|C_2|}{|C_1|+|C_2|}\overrightarrow{C_1C_2}$, ce qui donne alors :

$$\sum_{i\in C_1} r^2_{iC} = \sum_{i\in C_1} r^2_{iC_1} + \frac{|C_1||C_2|^2}{(|C_1|+|C_2|)^2} < \overrightarrow{C_1C_2}|\overrightarrow{C_1C_2} >= \sum_{i\in C_1} r^2_{iC_1} + \frac{|C_1||C_2|^2}{(|C_1|+|C_2|)^2} r^2_{C_1C_2} \quad (44)$$

De même nous pouvons calculer :

$$\sum_{i\in C_2} r^2_{iC} = \sum_{i\in C_2} r^2_{iC_2} + \frac{|C_2||C_1|^2}{(|C_1|+|C_2|)^2} r^2_{C_1C_2} \quad (45)$$

Et finalement :

$$\sum_{i\in C} r^2_{iC} = \sum_{i\in C_1} r^2_{iC} + \sum_{i\in C_2} r^2_{iC} = \sum_{i\in C_1} r^2_{iC_1} + \sum_{i\in C_2} r^2_{iC_2} + \frac{|C_1||C_2|}{|C_1|+|C_2|} r^2_{C_1C_2} \quad (46)$$

Nous en déduisons donc une expression simple de $\Delta\sigma$:

$$\Delta\sigma(C_1, C_2) = \frac{1}{n}\frac{|C_1||C_2|}{|C_1|+|C_2|} r^2_{C_1C_2} \quad (47)$$

A chaque étape, parmi les communautés ayant au moins une arête entre elles, nous choisissons donc de fusionner les deux communautés telles que la quantité $\Delta\sigma(C_1, C_2) = \frac{|C_1||C_2|}{|C_1|+|C_2|} r^2_{C_1C_2}$ soit minimale. Ceci ne demande aucun autre calcul que celui de la distance $r^2_{C_1C_2}$.

Un autre avantage est que le calcul des nouvelles variations de σ pour des fusions potentielles avec la nouvelle communauté $C_1 \cup C_2$ est très simple :

$$\Delta\sigma(C_1 \cup C_2, C_3) = \frac{(|C_1|+|C_3|)\Delta\sigma(C_1, C_3) + (|C_2|+|C_3|)\Delta\sigma(C_2, C_3) - |C_3|\Delta\sigma(C_1, C_2)}{|C_1|+|C_2|+|C_3|} \quad (48)$$

Cette propriété se montre (facilement mais de manière technique) en utilisant les propriétés des barycentre. Elle nous permet dans la plupart des cas de mettre à jour efficacement les nouvelles valeurs de $\Delta\sigma$. En effet nous connaissons déjà $\Delta\sigma(C_1, C_2)$, si de plus nous connaissions $\Delta\sigma(C_1, C_3)$ et $\Delta\sigma(C_2, C_3)$ alors nous obtenons directement $\Delta\sigma(C_1 \cup C_2, C_3)$ après fusion des communautés C_1 et C_2 en temps constant. Dans les autres cas nous devons recalculer une distance.

5.4 Fusion de communautés et mise à jour des données

Nous gardons en mémoire l'ensemble des communautés qui sont créées à chaque étape. Il y a n communautés au départ et il y a $n-1$ étapes créant chacune une communauté. Nous aurons donc exactement $2n-1$ communautés à stocker. Nous gardons en mémoire la liste des sommets ainsi que la liste des communautés voisines de chaque communauté faisant partie de la partition courante. Par contre pour les communautés n'en faisant plus partie (car elles ont été fusionnées avec une autre communauté), nous ne conservons que l'information donnant dans quelle nouvelle communauté elle a été fusionnée. Ceci permet de n'utiliser à tout instant

que $\mathcal{O}(n+m)$ espace mémoire et de garder une trace de toutes les étapes de fusion lors du déroulement de l'algorithme. Lors de l'analyse des résultats nous pourrons ainsi avoir accès à la composition de toutes le communautés moyennant quelques calculs supplémentaires.

Pour éviter de refaire des calculs inutiles, nous gardons en mémoire les vecteurs $P^t_{C\bullet}$ de chaque communauté C de la partition courante. Comme nous l'avons vu en 4.5.3, il est possible si nous ne disposons pas d'assez d'espace mémoire de ne pas stocker ces données ou de les stocker partiellement au détriment du temps d'exécution.

Nous conservons dans un arbre de recherche équilibré les valeurs $\Delta\sigma(C_1, C_2)$ entre toutes les communautés voisines (ayant au moins une arête entre elles). À tout instant il y a donc au plus m valeurs stockées dans cet arbre. Ainsi chaque ajout, suppression ou recherche du minimum se fait en un temps $\mathcal{O}(\ln m)$.

Lorsque l'on fusionne deux communautés C_1 et C_2 nous effectuons les étapes suivantes :

- Nous créons une nouvelle communauté dont la liste des sommets est la concaténation des deux listes de sommets de C_1 et C_2.
- Nous calculons ensuite $P^t_{(C_1\cup C_2)\bullet} = \frac{|C_1|P^t_{C_1\bullet}+|C_2|P^t_{C_2\bullet}}{|C_1|+|C_2|}$ et détruisons $P^t_{C_1\bullet}$ et $P^t_{C_2\bullet}$.
- Nous retirons de l'arbre toutes les valeurs $\Delta\sigma$ concernant C_1 ou C_2.
- Pour chaque voisin C_3 commun à C_1 et C_2 nous calculons et insérons dans l'arbre la nouvelle valeur de $\Delta\sigma(C_1 \cup C_2, C_3)$ avec la formule (48).
- Pour chaque autre voisin C_3 de C_1 ou C_2, nous calculons et insérons dans l'arbre la nouvelle valeur de $\Delta\sigma(C_1 \cup C_2, C_3)$ avec la formule (47).

5.5 Illustration du déroulement de l'algorithme sur un exemple

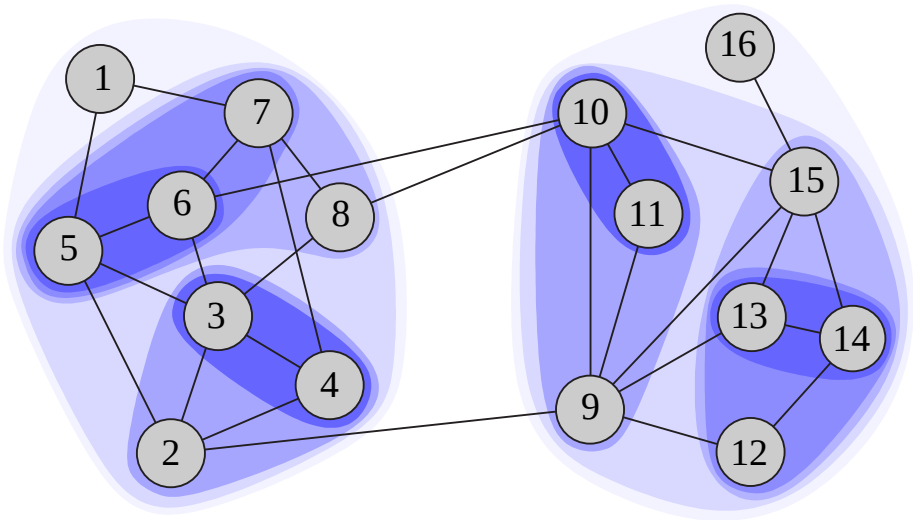


FIG. 3 – Exemple de structure de communautés trouvée par notre algorithme en utilisant des marches aléatoires de longueur $t = 3$.

Nous montrons ici le comportement de notre algorithme sur un petit exemple. La figure 3 montre le graphe utilisé et les différentes communautés trouvée. Nous avons effectué des calculs exacts avec des marches aléatoires de longueur $t = 3$. La figure 4 montre le dendrogramme correspondant à la structure hiérarchique des communautés obtenues. Chaque nœud de l'arbre

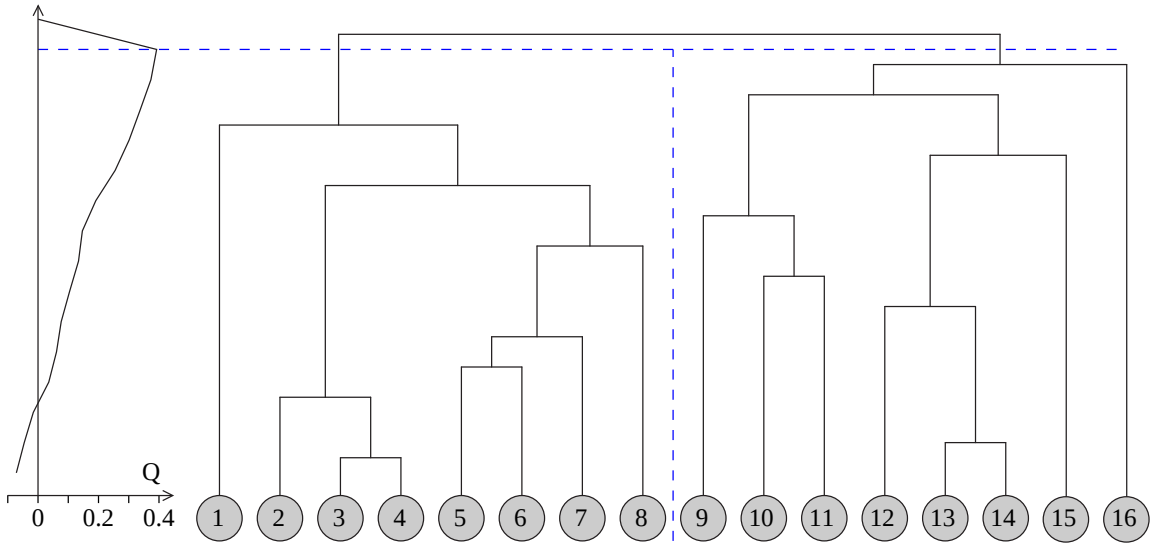


FIG. 4 – Dendrogramme correspondant à la structure hiérarchique des communautés trouvées par notre algorithme sur le graphe de la figure 3. À gauche : évolution de la modularité Q avec les fusions successives.

représente la fusion de deux communautés, l'ordre chronologique de ces fusions est codé par leurs hauteurs respectives. Le graphe de gauche représente la valeur de la modularité Q calculée après chaque fusion. La structure de communauté retenue par l'algorithme est celle correspondant à la valeur maximale de la modularité. Dans notre cas nous obtenons deux communautés, comme le montre le trait horizontal en pointillés.

5.6 Complexité globale

5.6.1 Cas des calculs exacts

Le calcul initial des probabilités $P^t_{i\bullet}$ pour tous les sommets i se fait en $\mathcal{O}(tmn)$ (ou en $\mathcal{O}(rmn)$ pour l'approche en temps continu). À chaque étape de fusion, le maintien de la structure de donnée codant les communautés se fait après chaque fusion en temps constant et le calcul de $P^t_{(C_1\cup C_2)\bullet}$ se fait en temps $\mathcal{O}(n)$. Ces deux étapes demandent donc pour les $n-1$ étapes un temps total en $\mathcal{O}(n^2)$.

La partie la plus coûteuse est le calcul des nouvelle valeurs de $\Delta\sigma(C_1\cup C_2, C_3)$. À chaque fusion, il faut recalculer les distances entre la nouvelle communauté créée et chacune de ses communautés voisines. Si C_3 était voisine des deux communautés C_1 et C_2, le calcul se fait en temps constant, sinon il se fait en $\mathcal{O}(n)$. Le paramètre jouant un rôle important dans la performance de l'algorithme est donc M, le nombre cumulé de communautés voisines de toutes les communautés créées au cours de l'algorithme. Le temps de calcul nécessaire est alors au plus $\mathcal{O}(Mn)$. Ceci correspond au cas défavorable où l'on ne fusionne jamais de communautés ayant au moins une même communauté voisine. Cependant dans la pratique la notion même de communauté a tendance à privilégier les cas favorables en fusionnant des communautés localisées dans des parties fortement clusterisées du graphe.

La valeur de M dépend en fait fortement de la profondeur H de la structure arborescente

hiérarchique des communautés créées. Nous pouvons définir la hauteur $h(C)$ de chaque communauté C de la façon suivante : les communautés initiales constituées d'un seul sommet sont à hauteur 0, la hauteur d'une communauté $C_1 \cup C_2$ est $h(C_1 \cup C_2) = \max(h(C_1), h(C_2)) + 1$. La profondeur H est la hauteur de la dernière communauté correspondant au graphe entier.

Une borne (très) supérieure simple de M est alors $2Hm$, en effet pour chaque hauteur $1 \leq h \leq H$, l'ensemble des communautés à hauteur h sont disjointes. La somme de leur nombre de communautés voisines est alors au plus $2m$ (chaque arête ne peut définir qu'une seule relation de voisinage entre deux communautés). La somme sur toutes les hauteurs possibles nous donne alors $M \leq 2Hm$. Nous pouvons noter ici que nous ne tenons pas compte du fait que plusieurs arêtes peuvent exister entre deux mêmes communautés et que de nombreuses arêtes se trouvent rapidement à l'intérieur d'une communauté. Dans les deux cas ce sont des arêtes qui sont comptées en trop dans notre borne supérieure.

Le pire des cas est $H = n - 1$. Nous obtenons donc une complexité de notre algorithme dans le pire des cas de $\mathcal{O}(mn^2)$. Ce pire des cas est réalisé dans le cas défavorable où une grande communauté intègre un à un des sommets isolés. Le graphe en étoile constitué d'un sommet central relié à $n - 1$ sommets de degré 1 est une illustration de ceci et atteint la complexité du pire des cas.

En pratique, l'algorithme de Ward que nous utilisons a tendance à créer de petites communautés de tailles équilibrées et s'approche du cas le plus favorable où la structure arborescente hiérarchique des communautés est un arbre équilibré. Dans ce cas favorable la profondeur est $H = \mathcal{O}(\ln n)$. Nous obtenons alors une performance de l'algorithme $\mathcal{O}(mn \ln n)$.

Rappelons encore une fois que toutes ces complexités supposent que les calculs des distances ne sont jamais faits en $\mathcal{O}(1)$ mais en $\mathcal{O}(n)$. En pratique cette optimisation apporte cependant un gain de performance appréciable.

5.6.2 Cas des calculs approchés

Nous avons vu que dans le cas des calculs approchés, le calcul de toutes les probabilités $P^t_{i\bullet}$ demande $\mathcal{O}(K(\ln K + t))$. Pour obtenir les probabilités $P^t_{(C_1 \cup C_2)\bullet}$, nous fusionnons les deux arbres $P^t_{C_1\bullet}$ et $P^t_{C_2\bullet}$. La taille des arbre de chaque communauté initiale étant au plus K, nous en déduisons que la taille de l'arbre stockant les $P^t_{C\bullet}$ pour une communauté C est au plus $|C|K$ et est de toute façon inférieure à n (lorsque nous approchons de n il est alors avantageux d'abandonner la structure d'arbre pour utiliser un tableau de taille n). La fusion des probabilités de deux communautés C_1 et C_2 se fait donc en $\mathcal{O}(\min(n, CK \ln(CK)))$.

Le calcul d'une distance entre deux communautés C_1 et C_2 se fera toujours au pire en temps $\mathcal{O}(n)$ mais peut être aussi fait en temps $\mathcal{O}((|C_1|+|C_2|)K)$ pour les petites communautés.

Ceci montre que dans tous les cas, le temps de calcul est toujours inférieur au temps de calcul de l'algorithme avec calcul exact.

6 Evaluation expérimentale des performances de l'algorithme

Il est difficile de tester un algorithme de détection de communautés car il faut pour cela posséder un jeu de graphes tests dont nous connaissons déjà la structure des communautés. Une approche répandue est l'utilisation de graphes générés aléatoirement possédant des communautés. Nous allons étudier dans cette partie de nombreux paramètres de notre algorithme en utilisant de tels graphes.

6.1 Graphes tests générés aléatoirement

6.1.1 Génération des graphes tests

Nous allons construire des graphes de n sommets possédant $c \geq 1$ communautés de tailles identiques $\frac{n}{c}$. Chaque sommet appartient à une communauté, son degré interne est le nombre de ses voisins faisant partie de la même communauté et son degré externe est le nombre de ses voisins faisant partie d'une autre communauté. Nous désirons générer de tel graphes possédant un degré interne z_{in} donné et un degré externe z_{out} donné. Pour cela nous partons d'un graphe à n sommet sans arêtes, nous attribuons ensuite chaque sommet à une communautés de sorte à créer c communautés de tailles égales. Nous tirons alors au hasard l'existence de chaque arête possible du graphe avec les probabilités adéquates :

- une arête reliant deux sommets d'une même communautés existera avec une probabilité $p_{in} = \frac{cz_{in}}{n-1}$
- une arête reliant deux sommets de deux communautés différentes existera avec une probabilité $p_{out} = \frac{cz_{out}}{n(c-1)}$

6.1.2 Evaluation de la performance d'un algorithme sur un graphe test

Nous attendons de notre algorithme qu'il retrouve les c communautés. Pour évaluer ses performances nous allons considérer le pourcentage η de sommets correctement identifiés. La définition de η est intuitive et non équivoque lorsque les communautés trouvées sont très proches des communautés réelles. Par contre lorsque les communautés trouvées ne correspondent pas aux communautés réelles il est plus difficile de le définir. Nous allons utiliser la définition suivante : Tout d'abord, nous allons identifier chaque communauté réelle C à la communauté trouvée $\widetilde{C}$ contenant le plus grand nombre de sommets de C. Si plusieurs communautés réelles sont identifiées à une même communauté trouvée $\widetilde{C}$ nous retiendront uniquement la communauté réelle ayant le plus de sommets dans $\widetilde{C}$. La valeur de η est alors le pourcentage de sommets correctement identifiés selon cette procédure.

6.2 Influence de la densité des communautés

Notre premier test général de performance de l'algorithme est effectué sur des graphes de 128 sommets possédant 4 communautés de 32 sommets. Nous effectuons des calculs exacts avec des marches de longueur $t = 4$. Pour évaluer sa capacité de détection des communautés, nous avons fait varier les paramètres z_{in} de 2 à 16 et z_{out} de 2 à 20 par pas de $0,25$. Pour chaque paire de paramètres, nous avons évalué la performance moyenne η de l'algorithme sur 100 graphes.

Nous pouvons observer que notre algorithme détecte de manière parfaite les communautés lorsque celles-ci sont nettes c'est à dire lorsque z_{in} est grand et z_{out} faible. Lorsque qu'au contraire z_{in} est faible et z_{out} grand, le graphe ne possède pas réellement de structure de communautés et il est donc normal d'obtenir des performances médiocres. Le taux η de reconnaissances de communautés qui sont en fait inexistantes ne représente donc rien de concret dans ce cas.

Les communautés sont donc toujours très bien détectées lorsqu'elles sont clairement identifiables. Lorsqu'elles deviennent plus diffuses notre algorithme obtient des résultats honorables qui sont meilleurs que ceux des différents algorithmes actuellement disponibles, présentés en 2.

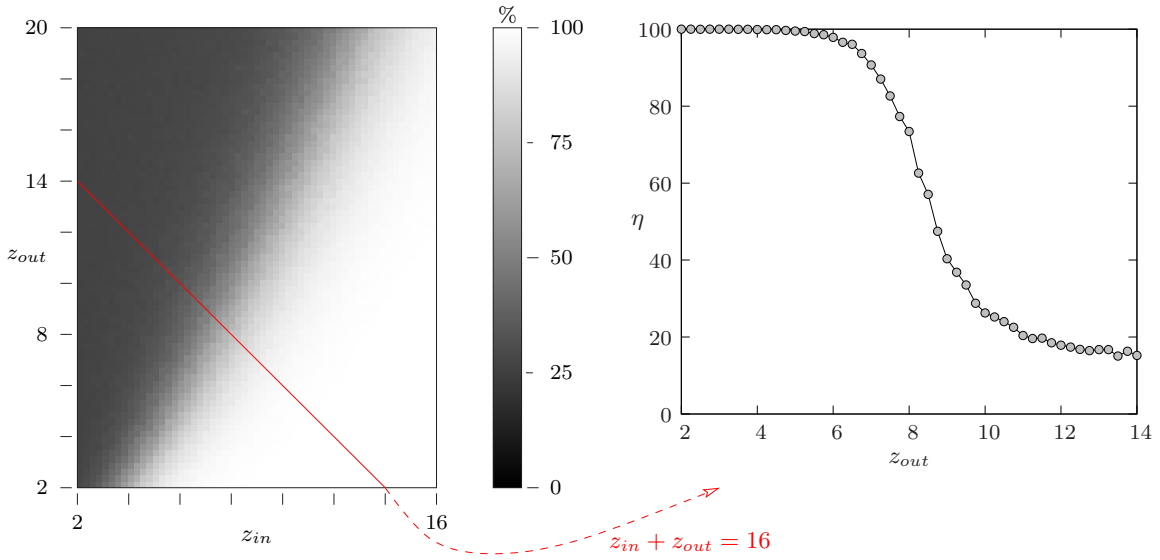


FIG. 5 – Gauche : pourcentage η de sommets correctement identifiés dans des graphes de 128 sommets possédant 4 communautés en fonction des degrés moyens internes z_{in} et externes z_{out} de chaque sommets. Droite : détail de cette courbe lorsque $z_{in} + z_{out} = 16$.

6.3 Influence de la longueur t des marches aléatoires

6.3.1 Sur les graphes tests simples

Nous avions remarqué dans 4.6 que la longueur t des marches aléatoires a une influence sur la taille des communautés détectées. Pour illustrer ceci, nous avons calculé pour différentes valeurs de t le taux de réussite sur des graphes tests de 64, 128 et 256 sommets comportant 4 communautés ($z_{in} = 9$ et $z_{out} = 7$). Nous avons choisi de ralentir la marche avec un paramètre $p = 0.5$ pour pouvoir apprécier plus de détails. Les résultats obtenus sont présentés dans la figure 6, chaque point du graphe est la moyenne de 1000 expériences.

Dans les trois cas, nous observons comme prévu que l'algorithme ne reconnaît pas efficacement les communautés pour des valeurs trop faibles de t car les marches aléatoires n'ont pas suffisamment de temps pour explorer les communautés entières. Les marches trop longues ne donnent pas non plus de bons résultats car elles atteignent le régime stationnaire et tous les sommets tendent à être considérés comme identiques. Nous pouvons cependant noter que les performances maximales sont atteintes pour des valeurs de t assez faibles. De plus, l'algorithme offre déjà des performances proches du maximum pour des valeurs de t très faible. Par contre, les performances décroissent lentement avec la longueur des marches lorsque celles-ci sont trop longues.

La valeur t_{max} pour laquelle les performances maximales sont atteintes est une fonction croissante de la taille des communautés à détecter (les valeurs de t_{max} pour les graphes de 64, 128 et 256 sommets sont respectivement 10, 14 et 16). Il en est de même pour la longueur de la plage de temps durant laquelle les performances sont proches du maximum.

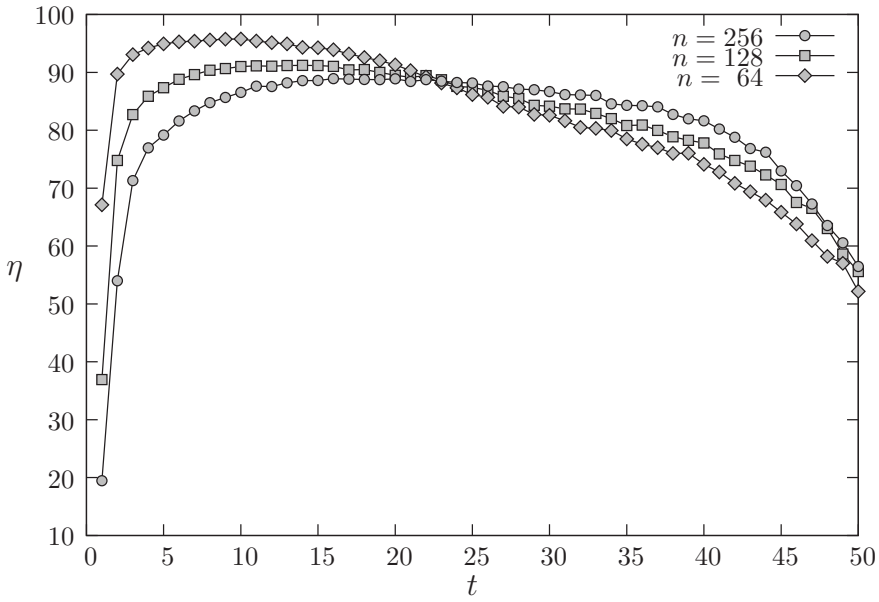


FIG. 6 – Pourcentage η de sommets correctement identifiés dans des graphes de 64, 128 et 256 sommets possédant 4 communautés ($z_{in} = 9$ et $z_{out} = 7$) en fonction de la longueur t des marches aléatoires effectuées (avec probabilité $p = 0.5$ de rester sur place à chaque pas)

6.3.2 Sur des graphes tests avec une structure hiérarchique de communautés

Nous proposons ici une seconde illustration de l'influence du choix de t sur de la taille des communautés détectées. Nous allons créer des graphes possédant une structure hiérarchique de communautés de la façon suivante : le graphe va posséder deux grandes communautés qui seront chacune divisées en deux communautés moyennes qui seront à leur tour composées de deux petites communautés (comme le montre la figure 7). Pour réaliser de tels graphes, chaque arête va être tirée au hasard avec une probabilité p_1, p_2 ou p_3 si elle est intérieure à une petite, moyenne ou grande communauté et avec une probabilité p_4 si elle est entre les deux grandes communautés.

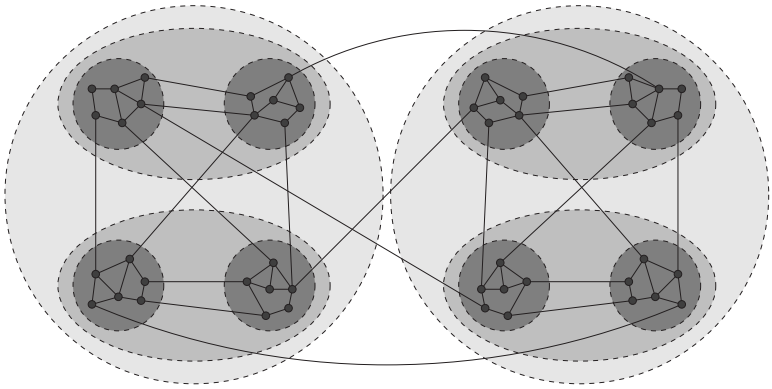


FIG. 7 – Structure hiérarchique des graphes utilisés pour nos tests

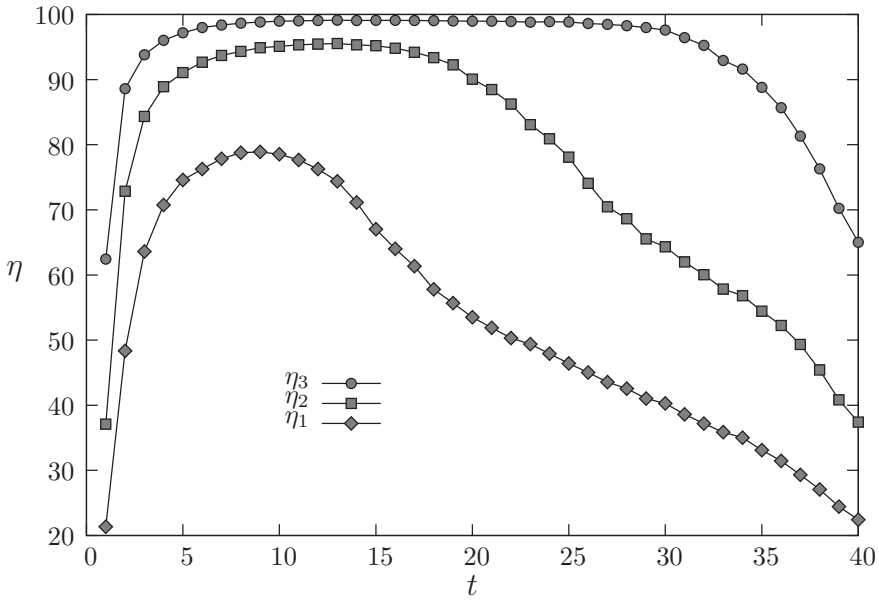


FIG. 8 – Pourcentage de sommets correctement identifiés dans les petites (η_1), moyennes (η_2) et grandes (η_3) communautés d'un graphe de 256 sommets possédant la structure hiérarchique décrite dans le texte en fonction de la longueur t des marches aléatoires ($p = 0.5$).

Nous avons choisi pour cette expérience des graphes de 256 sommets (chaque petite communauté contient 64 sommets). Les probabilités p_1, p_2, p_3 et p_4 ont été choisie de telles sortes que pour chaque sommet, les nombres moyens d'arête de chaque type soient respectivement $z_1 = 12$, $z_2 = 6$, $z_3 = 8$ et $z_4 = 12$. Nous avons calculé pour différentes valeurs de t (avec encore $p = 0.5$) les trois pourcentages η_1, η_2 et η_3 de sommets correctement identifiés dans les petites, moyennes et grandes communautés. Chaque valeur est la moyenne de 1000 graphes. Notons que l'on ne peu pas considérer ces valeurs au moment où l'algorithme décide de s'arrêter car les différentes communautés à distinguer se recouvrent. Nous avons donc évalué le taux de réussite pour chaque taille de communautés au moment où il est maximal, ce moment est différent pour chaque taille.

Les résultats (figure 8) confirment l'influence de la longueur des marches sur la taille des communautés détectées. Les petites communautés nécessites des marches de longueurs courtes pour être détectées et ne sont rapidement plus détectées lorsque t augmente. Les plus grandes communautés demandent des marches de longueurs un peu plus grandes pour être détectées de manière optimale et sont bien détectées sur un intervalle de t plus long.

6.4 Influence du calcul approché des probabilités P_{ij}^t

Notre dernier test vise à évaluer l'impact des calculs approchés sur les performances du l'algorithme. Nous utilisons encore des graphes de 128 sommets possédant 4 communautés ($z_{in} = 9$ et $z_{out} = 7$) avec des marches aléatoires ralenties ($p = 0.5$). Nous effectuons maintenant des calculs approchés avec différentes précisions correspondant à $K = 100$, $K = 1000$ et $K = 10000$ marches aléatoires simulées. Les résultats obtenus (figure 9) sont comparés avec ceux obtenus par des calculs exacts.

Nous observons que les calculs approchés permettent d'obtenir de bonnes performances

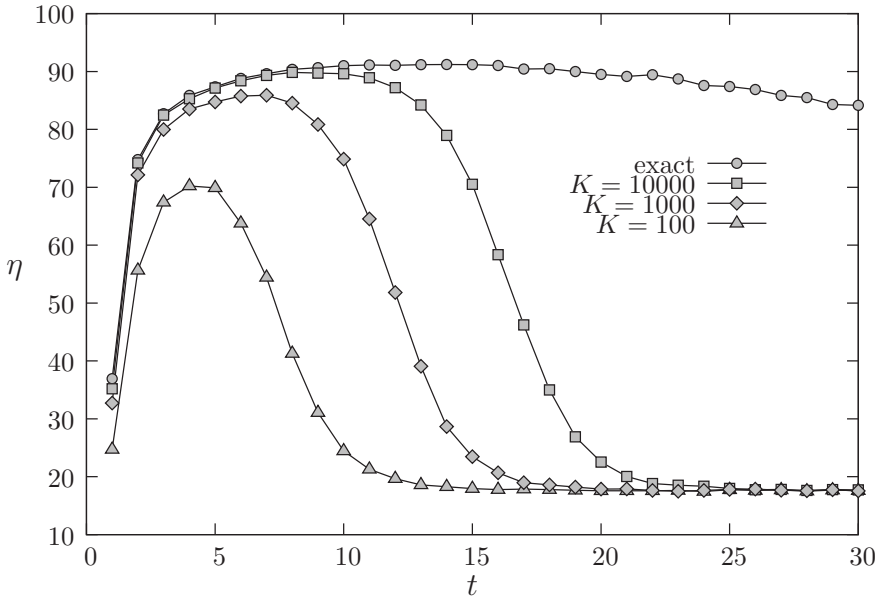


FIG. 9 – Pourcentage η de sommets correctement identifiés suivant la précision des calculs approchés pour des graphes de 128 sommets possédant 4 communautés ($z_{in} = 9$ et $z_{out} = 7$) en fonction de la longueur t des marches aléatoires effectuées ($p = 0.5$).

lorsque nous considérons des marches aléatoires courtes. Par contre, elles chutent très rapidement dès que la longueur des marches augmente. Ceci s'explique par le fait que les distances entre sommets tendent vers 0 lorsque t augmente. Les erreurs commises lors de l'estimation des probabilités P_{ij}^t deviennent alors trop importantes par rapport différences de valeurs à estimer. Ceci n'est pas trop contraignant car les performances de l'algorithme sont rapidement proches de l'optimal pour de faibles valeurs de t. Nous obtenons ainsi de très bonnes performances pour $K = 1000$ et $t = 6$.

Le tableau 2 compare les temps nécessaires au calcul des probabilités par les différents modes de calcul. Ces temps ne comprennent pas la phase de fusion, et l'implémentation actuelle de l'algorithme ne permet pas non plus d'exécuter cette phase pour les graphes de 100000 sommets par manque de mémoire. Les résultats montrent que les calculs approchés ne deviennent intéressants que pour de très grands graphes.

Mode de calcul	100 sommets	1000 sommets	10000 sommets	100000 sommets
Exact	8,9 ms	0,71 s	75 s	9 h
Approché $K = 100$	6,2 ms	74 ms	1,0 s	10 s
Approché $K = 1000$	41 ms	0,58 s	7,2 s	89 s
Approché $K = 10000$	370 ms	4,6 s	58 s	710 s

TAB. 2 – Temps (sur un P4-M 2.2Ghz avec 512Mo de Ram) nécessaire au calcul de toutes les probabilités suivant le mode de calcul et la taille du graphe. Les sommets ont un degré moyen de 15 et nous utilisons des marches aléatoires de longueur $t = 3$.

7 Conclusion

Nous avons proposé et implémenté un algorithme efficace de détection de communautés dans les grands réseaux d'interactions basé sur l'étude des marches aléatoires dans les graphes. Notre approche s'applique à des graphes de grandes tailles qui ne pouvaient pas être traités par la plupart des algorithmes existants. Ainsi, nous avons pu obtenir les structures de communauté de graphes de tailles allant jusqu'à 10 000 sommets, et nous estimons qu'il est possible de relever cette limite à 100 000 sommets en gérant la mémoire de manière plus astucieuse.

Les performances de notre algorithme permettent dores et déjà traiter des graphes réels de grande taille. Nous allons donc chercher à appliquer nos résultats à des problème réels ayant un finalité pratique et utile. Nous sommes aussi conscients que le choix de la longueur des marches aléatoires à utiliser est un problème qui n'est à ce jour que partiellement résolu. Un effort supplémentaire de recherche est donc à faire dans cette direction.

Nous n'avons considéré pour notre étude que des graphes non orientés, cependant certains graphes sont orientés (par exemple le graphe du Web). Nous pensons qu'il peut aussi être possible de sonder la structure de tels graphes à l'aide de marches aléatoires. L'étude mathématique sera moins aisée pour comprendre les nouveaux phénomènes mis en jeu. Par ailleurs, les communautés détectées par notre algorithme sont forcément disjointes. En réalité il se peut qu'un sommet appartienne à plusieurs communautés simultanément. À notre connaissance, il n'existe aucun algorithme permettant de détecter les structures de communautés qui se chevauchent. Nous estimons qu'il s'agit d'un élargissement naturel du problème méritant l'attention de futurs travaux.

Références

[1] S. Wasserman and K. Faust. *Social network analysis*. Cambridge University Press, Cambridge, 1994.

[2] Steven H. Strogatz. Exploring complex networks. *Nature*, 410 :268–276, March 2001.

[3] Reka Albert and Albert-Laszlo Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1) :47, 2002.

[4] M. E. J. Newman. The structure and function of complex networks. *SIAM REVIEW*, 45 :167, 2003.

[5] S.N. Dorogovtsev and J.F.F. Mendes. *Evolution of Networks : From Biological Nets to the Internet and WWW*. Oxford University Press, Oxford, 2003.

[6] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabasi. Hierarchical Organization of Modularity in Metabolic Networks. *Science*, 297(5586) :1551–1555, 2002.

[7] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans M. Coetzee. Self-organization and identification of web communities. *Computer*, 35(3) :66–71, 2002.

[8] Santo Fortunato, Vito Latora, and Massimo Marchiori. A Method to Find Community Structures Based on Information Centrality. *ArXiv :cond-mat/0402522*, February 2004.

[9] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 69(2) :026113, 2004.

[10] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Math. J.*, 23 :298–305, 1973.

[11] Alex Pothen, Horst D. Simon, and Kan-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3) :430–452, 1990.

[12] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2) :291–308, 1970.

[13] Mark S. Aldenderfer and Roger K. Blashfield. *Cluster Analysis*. Number 07-044 in Sage University Paper Series on Quantitative Applications in the Social Sciences. Sage, Beverly Hills, 1984.

[14] Brian S. Everitt, Sabine Landau, and Morven Leese. *Cluster Analysis*. Hodder Arnold, London, 4^{th} edition, 2001.

[15] Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301) :236–244, 1963.

[16] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *PNAS*, 99(12) :7821–7826, 2002.

[17] Ulrick Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2) :163–177, 2001.

[18] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *PNAS*, 101(9) :2658–2663, 2004.

[19] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 69(6) :066133, 2004.

[20] Luca Donetti and Miguel A. Muñoz. Detecting network communities : a new systematic and efficient algorithm. *arXiv :cond-mat/0404652*, 2004.

[21] Gaume Bruno. Balades aléatoires dans les petits mondes lexicaux. *I3 Information Interaction Intelligence*, (à paraître).

[22] L. Lovász. Random walks on graphs : a survey. In *Combinatorics, Paul Erdős is eighty, Vol. 2 (Keszthely, 1993)*, volume 2 of *Bolyai Soc. Math. Stud.*, pages 353–397. János Bolyai Math. Soc., Budapest, 1996.

[23] Ingve Simonsen, Kasper Astrup Eriksen, Sergei Maslov, and Kim Sneppen. Diffusion on complex networks : a way to probe their large-scale topological structures. *Physica A : Statistical Mechanics and its Applications*, 336(1-2) :163–173, May 2004.

[24] L. S. Schulman and Bernard Gaveau. Coarse grains : The emergence of space and order. *Foundations of Physics*, 31(4) :713–731, April 2001.

[25] Bernard Gaveau, Annick Lesne, and L. S. Schulman. Spectral signatures of hierarchical relaxation. *Physics Letters A*, 258(4-6) :222–228, July 1999.

[26] Roger B. Sidje and William J. Stewart. A numerical study of large sparse matrix exponentials arising in markov chains. *Computational Statistics & Data Analysis*, 29(3) :354–368, January 1999.