Stage de L3 : comparaison de communautés à l'aide de la modularité

Jonathan Protzenko

31 août 2008

Table des matières

PARTIE I
PRÉSENTATION

Ce stage s'est attaché à l'étude de l'environnement d'un site web : comment comprenons-nous l'ensemble des sites situés « autour » d'un site central ? Est-il possible de dégager des communautés de sites web, et si oui, comment ? Comment évaluer la qualité de ces communautés ? Ces questions ont fait l'objet d'une étude détaillée, à l'aide de deux approches distinctes que nous avons essayé de relier : la détection de communautés par la modularité, et l'algorithme HITS.

La détection de communautés par la modularité est une problématique récente, qui a réellement gagné en popularité avec l'article de Newman et Girvan en 2003 [NG04]. La présence de communautés au sein d'un réseau – social, par exemple – est un fait connu. L'apport de [NG04] est une mesure, la *modularité*, qui permet de quantifier la qualité d'un découpage en communautés. De nombreuses approches visant à maximiser cette qualité ont suivi ; nous en détaillons quelques-unes dans la partie suivante. Pour la détection de communautés à l'aide de la modularité, nous avons utilisé le meilleur algorithme à ce jour, décrit dans [BGLL08].

L'algorithme HITS ([Kle99]) est un algorithme qui cherche à utiliser l'information contenue dans les liens hypertextes pour en déduire les sites faisant autorité et les sites centraux : *authorities* et *hubs*. En effet, un lien d'un site vers un autre implique que le premier site reconnaît le second comme « intéressant », lui confère une valeur de « respectabilité » et d'autorité sur un sujet donné. Nous détaillerons plus précisément ces concepts dans la partie suivante. L'algorithme HITS a eu une très forte influence conceptuelle et a donné lieu à de nombreuses recherches sur des applications, améliorations, modifications possibles. Cette problématique n'est cependant plus vraiment d'actualité : les grands moteurs de recherche ont désormais leurs propres systèmes de notations pour les sites Web : PageRank pour Google, ExpertRank pour Ask.com…Même si HITS a inspiré certaines techniques concurrentes, aucun moteur de recherche ne proclame utiliser HITS tel quel. De plus, les secrets de fabrication d'algorithmes comme PageRank sont jalousement gardés, ce qui explique que la recherche dans ce domaine ne profite pas des avancées effectuées au sein des grandes entreprises. Enfin, l'algorithme HITS a été victime de son succès : de nombreux articles de qualité inégale ont été publiés dans son sillage, ce qui a contribué à mettre à un terme aux recherches sur ce sujet.

Au cours du stage, les deux algorithmes ont été testés, comparés, combinés sur le graphe de l'environnement d'un site web, et un nouvel algorithme a été proposé. Nous avons procédé de la façon suivante : après avoir défini précisément la question à laquelle nous tenterions de répondre (partie « Travail préliminaire »), nous proposons deux algorithmes (partie « Algorithmes développés »). Le premier est un algorithme naïf, et peu pertinent. Nous proposons alors un algorithme plus évolué, qui éclaircit la problématique initiale. Cependant, les données sont lourdement influencées, ainsi que nous le

verrons, par le poids des boucles : une deuxième étude (partie « Nouvelle étude sans les boucles ») permettra d'obtenir des résultats plus fins et plus pertinents. Enfin, la dernière partie revient de manière critique sur le travail effectué et propose quelques pistes pour des travaux futurs.

PARTIE II

RECHERCHE BIBLIOGRAPHIQUE ET ÉTAT DE L'ART

1 La littérature sur HITS

L'article fondateur [Kle99] a été publié en 1999. D'autres articles ont suivi, approfondissant le concept. En particulier, [GKR98] étudie en détail les communautés dégagées par l'algorithme HITS. On trouve également [KRRT99] qui, à partir des observations de Kleinberg[1], élabore une méthode pour détecter automatiquement dans le graphe des pages web des communautés en cours de constitution. Jon Kleinberg donne également quelques éclaircissements sur son algorithme HITS dans [CDG+99].

D'autres articles ayant eu moins d'impact existent dans le domaine. En effet, l'article fondateur [Kle99] a connu un succès considérable : de nombreux chercheurs se sont lancés dans des dérivés de HITS, ce qui a conduit à un florilège d'articles sur le sujet, de qualité inégale. Seuls les principaux sont cités ici.

Cette profusion d'articles explique également la baisse de popularité de HITS : après un succès considérable, et de nombreux articles sur le sujet, seuls des articles particulièrement novateurs sont susceptibles d'être publiés.

a. Présentation du fonctionnement de HITS

Nous résumons ici les explications données dans [Kle99], qui est l'article de référence concernant HITS. On pourra se référer à l'article original pour obtenir des informations plus détaillées.

HITS est conçu pour améliorer la qualité des moteurs de recherche. Son principe de fonctionnement est le suivant. Pour une recherche donnée, par exemple « jaguar », on collecte un ensemble de *pages web* de base en prenant toutes les pages contenant le terme « jaguar », ensemble noté S_0, auquel on ajoute l'ensemble des pages pointées par S_0 et l'ensemble des pages pointant vers S_0. L'ensemble ainsi obtenu est noté S.

Ainsi qu'expliqué dans [KL01], le graphe des pages web possède une structure particulière, à base d'*authorities* et de *hubs*. Les premières sont les pages qui font référence dans un sujet donné : en reprenant l'exemple ci-dessus, une *authority*[2] pour la recherche « jaguar » serait la page du constructeur automobile britannique. Une telle page possède naturellement de nombreux liens pointant vers elle, car elle est reconnue comme page de référence. La seconde catégorie regroupe les pages de « fans » : là où les authorities ne pointent que vers peu d'autres pages (le constructeur automobile jaguar, conscient de sa toute-puissance dans le domaine, aura tendance à faire peu de liens vers l'extérieur de son

[1] dont on pourra lire un résumé dans [KL01]

[2] Nous avons choisi de ne mettre en italique que les premières occurences d'*authority* : bien qu'« autorité » soit une traduction acceptable, hub ne possède pas de traduction évidente. Nous laissons donc ces termes dans la police normale désormais, afin de réserver l'italique aux mots mis en importance.

site), les hubs sont des pages qui pointent vers beaucoup d'autres pages, et sont elles-mêmes peu pointées. Autrement dit, un hub fait beaucoup de liens mais en reçoit peu. C'est typiquement un site de fan de voitures.

L'idée centrale dans l'algorithme HITS est que lorsqu'une page A fait un lien vers une page B, elle reconnaît à la page B une certaine autorité : elle lui confère une *valeur d'autorité*. Le principe de HITS consiste à donner à chaque page une valeur initiale d'authority et une valeur initiale de hub (on ne sait pas au départ si une page est plutôt hub ou authority), puis à mettre à jour, de manière itérative, les valeurs $x(s)$ et $y(s)$ (authority et hub) pour chaque page.

tant que *l'on souhaite itérer* **faire**
- ① $x(u) \leftarrow \sum_{(v,u)\in E} y(v)$;
- ② $y(u) \leftarrow \sum_{(u,v)\in E} x(v)$;
- ③ $\forall u, x(u) \leftarrow \frac{x(u)}{\sum_v x(v)^2}$;
- ④ $\forall u, y(u) \leftarrow \frac{y(u)}{\sum_v y(v)^2}$;

fin

Algorithme 1 : Algorithme HITS

Les différentes étapes de l'algorithme correspondent aux opérations logiques suivantes :

① la valeur d'authority d'une page est la somme des valeurs de hub des pages pointant vers elle ;

② la valeur de hub d'une page est la somme des valeurs d'authority des pages vers lesquelles elle pointe ;

③ et ④ il faut normaliser les valeurs d'authority et de hub pour chacune des pages.

Ainsi, si une page fait beaucoup de liens vers des pages avec une forte valeur d'authority, elle reçoit une forte valeur de hub. Réciproquement, si une page reçoit des liens de pages ayant des fortes valeurs de hub, elle reçoit une forte valeur d'authority.

Au bout de quelques itérations, les valeurs se stabilisent et l'on obtient un classement des sites permettant d'extraire les sites de référence sur un sujet donné, c'est à dire les sites les plus à-même de répondre au terme de recherche posé : ce sont ceux qui ont la plus forte valeur d'authority.

C'est ainsi qu'en 1999, HITS a permis d'exhiber pour la requête « moteur de recherche » les sites Altavista, Inktomi, etc. alors qu'auparavant, les moteurs de recherche classiques se contentaient de retourner l'ensemble des pages contenant le terme « moteur de recherche », et étaient incapables de se trouver eux-mêmes.

b. Justification mathématique de HITS

Si l'on note A la matrice d'adjacence du graphe[3], alors les deux premières lignes de l'algorithme 1 deviennent $x \leftarrow A^T y$ et $y \leftarrow Ax$. Pour deux itérations, on a alors : $x \leftarrow (A^T A)x$ et $y \leftarrow (AA^T)y$ où A^T dénote la transposée de A. Ainsi, lorsque les itérations successives n'apportent plus de modifications, et qu'on a obtenu les valeurs finales de hub et d'authority pour chaque page de S, on a $x = x^*$ avec x^* tel que $(A^T A) \times x^* = \lambda x^*$. De même[4], on a[5] $y = y^*$ avec $(A^T A)^T \times y^* = \lambda y^*$.

En fait, le processus itératif que nous venons de décrire est la méthode de calcul du vecteur propre (dans notre cas, x ou y) associé à la plus grande valeur propre (λ) par itération de puissance (*power method* en anglais). Le processus converge sous certaines conditions[6].

Les itérations de $(A^T A)$ convergent vers le vecteur propre x^* de cette même matrice et les itérations de (AA^T) convergent vers le vecteur propre y^* de cette dernière matrice. Ces deux vecteurs sont associés à une même valeur propre λ. Nous ne nous intéresserons dans la suite qu'au vecteur propre x^* qui représente les valeurs d'authority.

Dans notre cas, les différentes conditions de convergence sont assurées, nous pouvons donc affirmer que le processus converge bel et bien vers des valeurs de hub et d'authority. On pourra au besoin consulter l'excellent [AK08], qui aura le double avantage de constituer un excellent rappel des notions principales d'algèbre linéaire, et de détailler avec preuve la méthode de calcul du plus grand vecteur propre (appelé vecteur propre principal, par opposition aux vecteurs propres non-principaux) par itération de puissance, ainsi que d'autres méthodes utiles par la suite.

On notera également que le vecteur propre ainsi obtenu ne possède que des composantes de même signe (on prendra positif comme convention).

c. Inteprétation avancée de HITS : présence de communautés

Jon Kleinberg, après avoir présenté dans [Kle99] les principes fondamentaux de son algorithme, a également introduit la présence de communautés dans ce même article. On retrouve des explications supplémentaires dans [CDG+99].

Intuitivement, pour les vecteurs propres étoilés cités ci-dessus, les valeurs de hub et d'authority assignées aux pages sont *stables* : on est dans une situation où les valeurs de hub et d'authority se confortent mutuellement. Que signifient alors les autres vecteurs propres, dits non-principaux ?

Tout d'abord, ils possèdent des composantes de signes différents. Contrairement à ce que l'on pourrait penser à première vue, ceci s'interprète très précisément : des

[3] $a_{ij} = \begin{cases} 1 & \text{si arête entre i et j} \\ 0 & \text{sinon} \end{cases}$

[4] Le λ est le même car les *valeurs* propres d'une matrice et de sa transposée sont identiques (se montre avec le polynôme caractéristique, par exemple).

[5] On renormalise une dernière fois x^* et y^*.

[6] En fait, le processus converge si :
- la matrice est diagonalisable (c'est le cas, $A^T A$ est symétrique réelle, donc autoadjointe, donc diagonalisable, cf. théorème spectral)
- ses valeurs propres sont réelles (même argument)
- la valeur propre avec le plus grand module est simple (son espace propre est de dimension 1) et positive
- dans la base des vecteurs propres, le vecteur initial (les valeurs initiales de hub et d'authority choisies pour les pages) s'écrit $\sum_{i=1}^{n} \beta_i e_i$ avec $\beta_n \neq 0$.

De plus, la vitesse de convergence est un $\mathcal{O}\left(\frac{|\lambda_{n-1}|}{\lambda_n}\right)$ où les λ_i sont les valeurs propres classées dans l'ordre croissant : $|\lambda_1| \leqslant \cdots \leqslant |\lambda_{n-1}| < \lambda_n$.

pages de même signe se voient placées dans une relation de renforcement mutuel. C'est-à-dire qu'on a des sous-ensembles de pages qui valident la relation de hub et d'authority, de manière d'autant plus forte que la valeur propre associée au vecteur est importante. On a ainsi une partition en deux, selon le signe des composantes du vecteur x_i^*, des pages. Pour chacun des deux sous-ensembles, les pages possédant la plus forte valeur d'authority sont les sites les plus pertinents sur un des deux sous-sujets qui ont été dissociés dans ce vecteur. Ainsi, en étudiant les plus fortes authorities pour chaque signe des premiers vecteurs propres non-principaux, on distingue par exemple les pages faisant autorité sur l'animal jaguar, et les pages faisant autorité sur la marque automobile. Ceci n'était pas possible pour le vecteur propre principal puisque celui-ci classait toutes les pages comme s'ils faisaient partie d'une même communauté.

Les pages extraites par HITS cessent cependant rapidement d'être pertinentes au sens d'un sujet précis, dans la mesure où les λ_i décroissent rapidement : seuls les vecteurs propres associés à de fortes valeurs propres portent une relation significative dans leur classement. Nous détaillerons plus loin ce que nous entendons par « fortes valeurs propres ».

Enfin, nous ne nous intéressons que peu aux hubs : en fait, même s'ils sont importants dans le processus itératif de l'algorithme, au niveau sémantique, ils ne sont pas susceptibles de répondre précisément aux termes de recherche soumis. Par exemple, on classera facilement dans la catégorie hubs un blog, qui fait beaucoup de liens vers l'extérieur mais en reçoit peu. C'est typiquement le genre de site dont il est peu probable qu'il réponde à la question posée.

2 La littérature sur la détection de communautés

La détection de communautés est une problématique assez ancienne, qui tire ses origines dans les problèmes de partitionnement de graphe, et entretient des liens forts avec la sociologie. Cependant, dans la dernière décennie, des communautés sont apparues naturellement dans de nombreux réseaux pratiques : pages web, mais également réseaux d'interaction de protéines, de gènes, réseaux sociaux type Facebook, réseaux pair à pair, topologie d'Internet… L'étude de ces réseaux fait partie du domaine des systèmes complexes.

Dès lors, il a été nécessaire d'élaborer des algorithmes capables de dégager des communautés sans connaître au préalable leur nombre, tout en cherchant le « meilleur découpage en communautés ». L'article fondateur est dû à Girvan et Newman [NG04] : il propose une première méthode de découpage en communautés, tout en introduisant une mesure pour quantifier la qualité d'un découpage : la *modularité*.

La détection de communautés est un domaine de recherche extrêmement actif et fécond. À l'heure actuelle, il est possible de détecter des communautés dans des réseaux de plusieurs centaines de millions de nœuds, tout en dégageant plusieurs niveaux hiérarchiques de découpage en communautés, en utilisant [BGLL08]. La course à des algorithmes toujours meilleurs continue : comme trouver une communauté optimale est un problème NP-complet, les algorithmes cherchent constamment une meilleure approximation.

Certaines problématiques n'en sont encore qu'à leurs balbutiements : on ne comprend encore pas bien les communautés qui se chevauchent, les communautés dans des graphes orientés, ou l'évolution des communautés dans un graphe dynamique. Les algorithmes comme [BGLL08] ne proposent aucune réponse à ces problèmatiques.

a. Les différentes méthodes de détection de communautés

Différentes méthodes ont été proposées. La première [NG04] utilise la *edge betweenness* : pour une arête donnée, on compte le nombre de plus courts chemins entre deux sommets passant par cette arête. Si ce nombre est élevé, c'est que cette arête est probablement un point de passage obligé pour aller d'une partie du graphe vers une autre, c'est à dire qu'elle relie deux communautés. Par élimination successive d'arêtes, on arrive à séparer les différentes communautés du graphe. Cette méthode est cependant extrêmement coûteuse. Ont ensuite suivi différentes techniques : méthode agglomérative en fusionnant des communautés [New04, CNM04], valeurs propres [New06], marches aléatoires [PL05]. D'autres méthodes incluent par exemple le recuit simulé (coûteux, rapidement abandonné). Des améliorations [WT07] portant sur les structures de données utilisées ont permis, en accélérant les performances du code, d'étudier des réseaux sociaux type Facebook de petite taille (quelques millions d'utilisateurs). Enfin, [BGLL08] est l'algorithme qui permet, à l'heure actuelle, de traiter les plus grands réseaux.

[BGLL08] fonctionne de la façon suivante : pour chaque nœud u, et pour chacun des voisins v de ce nœud, on regarde le gain de modularité obtenu en mettant u dans la communauté de v. S'il existe un voisin v tel que mettre u dans la communauté de v provoque une augmentation de la modularité, alors on met u dans la communauté de v^* avec v^* celui d'entre les voisins tel que le gain de modularité est maximal. Tant que, parmi tous les nœuds, il existe une opération de la sorte qui fait gagner de la modularité, on continue d'agglomérer les nœuds. Une fois qu'il n'est plus possible d'augmenter la modularité, on reprend du début, mais pour le méta-graphe, c'est à dire le graphe dont les nœuds sont les communautés, et les arêtes les poids des liens entre les communautés[7]. Plusieurs méta-graphes peuvent ainsi être construits au cours de l'exécution de l'algorithme.

On peut considérer que la détection de communautés en cherchant à maximiser la modularité fait consensus. Néanmoins, cette méthode a ses faiblesses : outre le biais de la mesure (voir [FB06], qui étudie de ma-

[7] L'étape de passage aux méta-communautés n'a que très rarement été observée dans notre cas : sur tous les exemples où nous avons exécuté BGLL, les passes (nouvelles exécutions sur le méta-graphe) supplémentaires n'ont apporté aucun gain de modularité. Les auteurs de l'algorithme expliquent en effet que les performances de BGLL sont très bonne dès la première passe, et que sur de petits graphes, des passes supplémentaires n'ont pas d'intérêt en termes de modularité.

nière très rigoureuse les biais de la modularité : il montre en particulier que la modularité dépend de la taille du graphe, et qu'ainsi certaines communautés trop petites ne sont pas récompensées par la modularité (d'où le titre : « Resolution limit in community detection »). [FB06] montre également comment construire artificiellement un graphe de modularité maximale.), cette modularité ne prend aucunement en compte le cas où les communautés se chevauchent. Ceci a conduit des auteurs à essayer d'autres méthodes, comme par exemple la détection locale [LWP06] sans introduire la modularité. On notera également la détection de communautés qui se superposent à l'aide de la percolation de cliques [PFP+07].

b. Une mesure : la modularité

Pour mesurer la qualité d'un découpage en communautés, une mesure a été introduite [NG04] : la modularité. Posons, pour un nœud v, c_v la communauté de v. Posons $m = \frac{1}{2}\sum_{vw} A_{vw}$ où A_{vw} vaut 1 s'il existe une arête entre v et w, 0 sinon. L'expression de la modularité la plus facilement manipulable est

$$Q = \sum_i e_{ii} - a_i^2$$

où

$$e_{ij} = \frac{1}{2m}\sum_{vw} A_{vw}\delta(c_v, i)\delta(c_w, j)$$

ou de manière plus intuitive : e_{ij} est le poids (normalisé) des arêtes entre la communauté i et la communauté j. Alors

$$a_i = \frac{1}{2m}\sum_v k_v \delta(c_v, i)$$

où $k_v = \sum_w A_{vw}$, ou formulé autrement, k_v est le degré de v. a_i est donc le poids total (normalisé) des arêtes connectées à des nœuds de la communauté i.

Cette écriture de la modularité permet en fait de concrétiser une définition intuitive d'une communauté : une communauté est un ensemble de nœuds pour lesquels la densité d'arêtes est plus forte à l'intérieur de la communauté qu'à l'extérieur.

Cette modularité peut également s'exprimer en termes de graphes aléatoires[8] : elle vaut alors

$$\frac{1}{2m}\sum_{vw}\left[A_{vw} - \frac{k_v k_w}{2m}\right]\delta(c_v, c_w)$$

De manière intuitive, le terme $\frac{k_v k_w}{2m}$ est le nombre moyen d'arêtes entre v et w [9] qu'il existe une arête entre v et w. La modularité mesure donc la différence de densité entre les nœuds d'une même communauté entre le graphe considéré et un graphe aléatoire. Autrement dit, si le graphe était aléatoire, la modularité serait nulle[10]. Plus la modularité est forte, plus la structure en communautés est marquée, et s'éloigne de l'aléatoire.

[8] se référer à [CNM04] pour l'équivalence, on y trouvera la meilleure et la plus propre définition de la modularité

[9] Ce nombre moyen ne s'entend pas au sens du modèle de graphes aléatoires d'Erdös-Renyi : dans le modèle ici considéré, on conserve la distribution des degrés.

[10] Cela n'exclut aucunement une modularité négative.

PARTIE III
TRAVAIL PRÉLIMINAIRE

1 La question étudiée – motivations

Les communautés de HITS sont-elles de bonnes communautés au sens de la modularité ?

Ce stage trouve son origine dans une interrogation, à première vue éloignée de la question formulée ci-dessus : existe-t-il des communautés de sites web ? Cette question, posée naïvement, en toute ignorance de HITS ou de BGLL, allait cependant donner naissance à la problématique qui est au cœur de ce stage.

Une première piste de recherche s'est constituée autour des algorithmes de détection de communauté et leurs applications au graphe des sites web. Une première implémentation de l'algorithme BGLL a permis de lancer de la détection de communautés dans l'environnement d'un site web, concrétisant ainsi le terme de « communauté de sites web ».

Cependant, la détection de communautés de BGLL produisait parfois des résultats surprenants, ce qui nous a conduits à questionner la légitimité de ces communautés. L'algorithme HITS est alors entré en ligne de compte : universellement reconnu, c'était en quelque chose le point de référence qui allait nous servir d'étalon pour juger de la qualité d'éventuelles autre méthodes.

La question s'est alors posée autrement : puisque les communautés dégagées par l'algorithme HITS sont, par nature, pertinentes pour le graphe des sites web, si la modularité est une mesure adaptée à la détection de communautés web, alors elle doit récompenser HITS en « notant » favorablement les communautés dégagées par ce dernier.

Ainsi est venue la question : les communautés de HITS sont-elles de bonnes communautés au sens de la modularité ? Nous tenterons de répondre de diverses manières à cette question dans les pages qui suivent.

2 Sur quel graphe travailler ?

L'objet du stage était l'étude de l'environnement d'un site web. Avant toutes choses, il faut donc récupérer cet environnement : la procédure retenue est la suivante (exécutée un nombre fini d'itérations).

i) initialement, $S = \{\text{page de départ}\}$;

ii) on pose T = l'ensemble des url pointées par S et pointant[11] vers S ;

iii) on récupère les pages étant dans T ;

iv) on pose $S = T$ et on reprend à l'étape ii)

L'ensemble des pages récupérées au cours de l'algorithme constitue l'environnement du site web (l'union des S successifs). Dans la plupart des cas, trois itérations

[11] On récupère ces dernières en interrogeant Google avec `link:`. À cause du très grand nombre de pages pouvant pointer sur des sites tels `http://www.cnrs.fr`, nous n'en récupérons que les n premières, typiquement, 20

de l'algorithme suffisent pour avoir des données intéressantes.

Un problème se pose cependant : si l'on crée le graphe où chaque nœud représente une *page web*, la détection de communautés donne des résultats peu intéressants. Les communautés détectées sont, à peu de choses près, les *sites web* : en effet, grâce aux nombreux menus de navigations, liens internes, plans de site, etc., l'ensemble des pages d'un site web constitue une communauté en elle-même dans le mesure où les liens internes à un site web sont très nombreux. La détection de communautés perd donc de son intérêt.

Pour continuer une étude qui ait du sens, c'est le graphe des *sites web* qui a été retenu : un nœud symbolise un nom de domaine [12] : `http://www.ens-lyon.fr` par exemple. Toutes les pages dont l'URL commence par `http://www.ens-lyon.fr` seront assimilées au nœud `http://www.ens-lyon.fr`[13]. Le nœud aura donc pour arêtes les arêtes de toutes les pages qu'il représente. Chaque lien entre deux *pages* compte pour un poids de 1 : on trouve donc des poids de l'ordre de quelques dizaines, typiquement, entre deux *sites*, car les poids des liens de leurs pages s'additionnent. C'est donc un graphe orienté et avec des poids sur les nœuds que nous créons. Notons enfin que ce graphe possède naturellement des boucles, représentant tous les liens internes à un site.

L'algorithme HITS sera donc lancé sur le graphe des *sites web* et non pas des pages web. Conceptuellement, l'idée est la même et les explications restent valables. Simplement, on travaille à une échelle plus grande que l'algorithme initial. On prend également en compte les poids des arêtes : il suffit de remplacer les 1 dans la matrice d'adjacence par le poids des arêtes. Là encore, on ne change pas fondamentalement l'algorithme : nous l'adaptons juste aux réalités du Web actuel, qui a largement évolué depuis 1999.

3 Progression

La mise en place des outils nécessaires à une étude rigoureuse et efficace a impliqué des difficultés techniques sur lesquelles ne nous étendrons pas. Nous retiendrons que des outils de calcul scientifiques ont été utilisés pour les implémentations en Python des différents algorithmes : NumPy et SciPy pour la manipulation de données, Matplotlib pour les graphiques. Pour l'extraction des valeurs propres de la matrice de HITS, une bibliothèque performante a été utilisée : ArPACK (`http://www.caam.rice.edu/software/ARPACK/`) implémentant une version modifiée de l'algorithme de Lanczos et de la méthode QR que l'on pourra étudier avec délectation dans [AK08].

[12] `ens-lyon.fr`, `ixxi.fr` et ainsi de suite ; `www.ens-lyon.fr` est un sous-domaine qui fait partie du domaine `ens-lyon.fr`

[13] Un raffinement que nous avons pris en compte est les URL du type `http://www.example.com/~user`. De toute évidence, c'est un site à part entière qu'il faut voir dans cet URL. Le représentant sera donc `http://www.example.com/~user` pour toutes les pages commençant ainsi. Pour les autres pages, du type `http://www.example.com/sthelse/`, le représentant reste `http://www.example.com`. Nous n'avons pas poussé les raffinements plus loin : il aurait fallu ajouter pour `http://perso.ens-lyon.fr/` un comportement particulier, mais il n'existe pas de moyen général de le détecter.

PARTIE IV
ALGORITHMES DÉVELOPPÉS

L'idée est d'étudier les communautés données par HITS à l'aide de la modularité. Mais quelles communautés ? Nous avons décrit dans les grandes lignes ce que l'on entendait par « communautés de HITS » dans la partie précédente. Nous allons nous intéresser à la manière de construire de telles communautés, et aux résultats que l'on obtient pour les communautés ainsi choisies.

Deux algorithmes sont détaillés dans cette partie : l'algorithme 2 et l'algorithme 3. Pour chacun des deux algorithmes, nous commençons par décrire leur fonctionnement. Les résultats sont ensuite analysés à l'aide de deux approches : une approche qualitative, qui consiste à étudier précisément le découpage en communautés, en regardant quels sites sont regroupés dans une même communauté. Le critère intuitif pour juger de ce découpage est la pertinence sémantique du regroupement entre sites : nous considérerons que l'algorithme a été efficace s'il regroupe des sites parlant du même sujet. La seconde approche consiste à utiliser la modularité comme mesure objective permettant de juger de la qualité d'un découpager. On s'intéresse cette fois-ci plus à l'aspect général du découpage : en regardant le graphe, a-t-on l'impression que les communautés que l'on observe (des ensembles de nœuds denséments liés) sont celles choisies par l'algorithme ? Nous admettrons ici généralement que le découpage proposé par [BGLL08] est identifiable avec le découpage en communautés fournissant la meilleure modularité.

On utilisera donc indifféremment « découpage de BGLL », « découpage au sens de la modularité » pour parler du découpage obtenu par l'algorithme BGLL et donnant (à peu de choses près) la meilleure modularité. Par opposition, on parlera du « découpage de HITS » pour parler du découpage donné par l'un des deux algorithmes développés durant ce stage.

1 Les données utilisées

Nous avons lors de l'étude travaillé avec les sites suivants : Suez (`http://www.suez.fr`), Veolia (`http://www.veolia.fr`), Ixxi (`http://www.ixxi.fr`), Ens-Lyon (`http://www.ens-lyon.fr`), serveur des pages perso de l'Ens-Lyon (`http://perso.ens-lyon.fr`), le site personnel de l'auteur (`http://www.xulforum.org`), un blog parmi les plus lus en France (`http://www.embruns.net`), un autre blog très lu (`http://www.maitre-eolas.fr`), le portail des développeurs XUL francophone (`http://www.xulfr.org`), Mozilla.org (`http://www.mozilla.org`), pages personnelles d'Yves Robert, de Daniel Hirschkoff, d'Eddy Caron, page listant les membres de Graal…

Nous n'avons fait figurer en exemple dans ce rapport que trois sites : Ixxi, Mozilla, et Suez. Nous avons en effet rencontré trois grands comportements lors du parcours de l'environnement d'un site web :

– certains sites possèdent un environnement très peu varié : dans le cas des sites d'entreprise (Suez, Veolia), l'intégralité des liens sont des liens internes au domaine (`.suez.fr` par exemple). Ces sites possèdent un grand nombre de pages et une très forte densité de liens internes. Même avec une profondeur trois, on ne s'éloigne que très peu du domaine initial. De plus, Google `link:` ne retourne lui-même que des pages du domaine initial. Nous avons retenu pour ce types de sites **Suez**, sur lequel nous avons opéré un parcours de profondeur 4, afin de ne pas obtenir que des pages du domaine `.suez.fr`.
– À l'opposé, d'autres sites emmènent, dès la première étape du parcours, vers des endroits très divers de la toile : c'est le cas des blogs que nous avons étudiés. Dans le cas d'Embruns, un simple parcours de profondeur 3 a duré plusieurs jours, tant le nombre de sites dégagés était important. La puissance de calcul a d'ailleurs manqué pour analyser ce graphe de près de 100000 nœuds[14]. Nous avons donc retenu pour représenter ce type de sites le site de **Mozilla**. C'est un bon choix car Mozilla ne possède pas dans son environnement de sites d'entreprise (qui donc nous emmèneraient dans les méandres de leurs liens internes) ; néanmoins, des sites très variés ont été dégagés dans l'environnement de ce site.
– Enfin, un type de site intermédiaire concerne les sites académiques : pages personnelles de chercheurs, sites de laboratoires. Dans l'environnement de ces sites, on retrouve des sites institutionnels (le CNRS par exemple) qui présentent les caractéristiques des sites d'entreprise, mais aussi des pages personnelles de chercheurs, ou des pages sur un sujet très précis, qui permettent de découvrir rapidement d'autres pages. C'est donc un mélange des deux types de sites présentés ci-dessus. L'**Ixxi** a été retenu pour représenter ce type de sites.

Lors des parties visant à illustrer les propos théoriques, nous ne présentons en exemple que deux des trois sites retenus ci-dessus. Ce nouvel élagage peut sembler surprenant. Néanmoins, comme nous le constaterons par la suite, les conclusions dégagées sont largement similaires. Il nous a dès lors paru judicieux de ne pas surcharger le lecteur avec des exemples répétitifs.

2 Algorithme naïf (algorithme 2)

L'algorithme naïf (algorithme 2) consiste naturellement à parcourir les vecteurs propres non-principaux contenant les valeurs d'authority : on sait que pour chaque vecteur, si l'on trie les sites par leur valeur d'authority, les premiers sites sont les plus pertinents pour un sous-sujet donné, de même que les sites avec les plus fortes valeurs négatives sont les plus pertinents pour un autre sous-sujet donné.

Plusieurs problèmes se posent alors.

[14] L'ordinateur personnel (vieux) de l'auteur ayant servi aux calculs, il a fallu se limiter à des graphes permettant d'obtenir des résultats en un temps raisonnable, soit quelques heures. Même avec les machines du LIP, l'extraction des vecteurs propres aurait pris plusieurs jours, car c'est une opération extrêmement coûteuse. Les données sont précieusement gardées pour des études futures

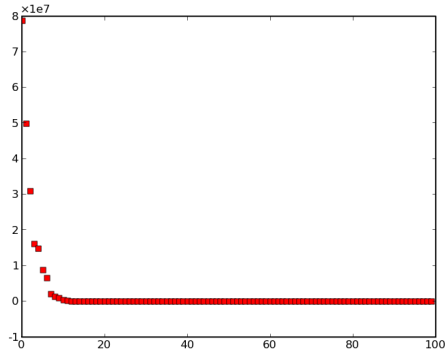

FIG. 1 – Décroissance des valeurs propres pour les 100 premiers vecteurs propres (cas de l'IXXI)

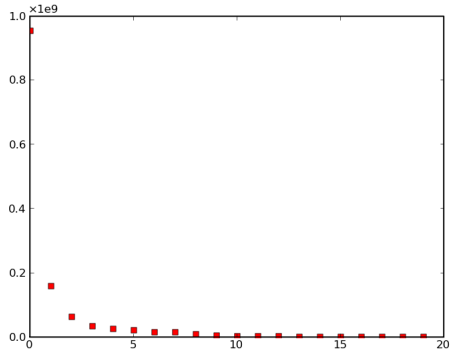

FIG. 2 – Décroissance des valeurs propres pour les 20 premiers vecteurs propres (cas de Mozilla)

- Doit-on essayer de trouver une communauté pour chaque site ou est-il préférable de construire un sous-graphe du graphe de départ ne contenant que les sites pour lesquels HITS a détecté une forte appartenance à une communauté ?
- Si l'on choisit de ne garder que les sites les plus « communautarisés », combien de vecteurs propres doit-on garder ? Quel est le critère nous informant que les valeurs propres sont « trop faibles » et que les relations de hub et d'authority ne sont plus assez marquées pour constituer des communautés pertinentes ?
- Pour un vecteur donné, combien de sites doit-on garder comme étant pertinents par rapport au sous-sujet isolé ? De même, existe-t-il un critère nous informant que la valeur d'authority est trop faible et que le site ne doit plus être considéré comme faisant partie du sous-sujet ?

Pour ce qui est de l'interrogation première, nous avons choisi de ne garder qu'un sous-graphe du graphe de départ. Cette approche, plus féconde, se justifie par le fait que certains sites sont tout à fait périphériques, et que même si le parcours de l'environnement d'un site permet la plupart du temps d'exhiber des sites importants, il reste malgré tout des sites d'un intérêt limité. Les éliminer n'est en aucun cas une solution « par défaut » ou un processus arbitraire. Le fait même que HITS ne les place dans aucune des communautés principales prouve justement qu'ils ne possèdent que peu d'intérêt. Il apparaît alors comme légitime de les écarter du sous-graphe communautaire extrait du graphe initial.

Avant d'apporter des réponses pour les deux interrogations restantes, considérons les graphiques 1 et 2.

Le profil de courbe observé dans ces deux graphiques a été constaté dans tous les autres sites que nous avons observés. La décroissance rapide s'accompagne d'un aplatissement considérable. Lorsqu'on sait que le graphe de Mozilla possède 2700 nœuds et 27000 arêtes, que celui de l'IXXI possède 500 nœuds, et 1500 arêtes, il est surprenant de constater que seuls quelques sous-sujets ont été mis en évidence par HITS. C'est néanmoins une caractéristique avec laquelle il faudra composer : seules quelques communautés sont détectées par HITS.

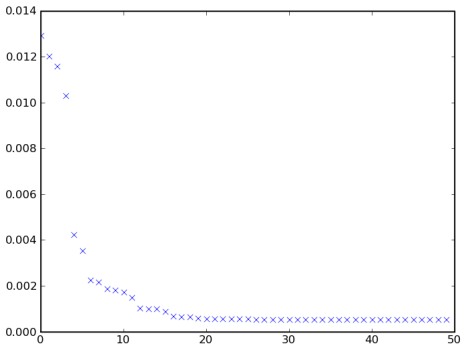

FIG. 3 – Décroissance des valeurs d'authority pour le 3^{ieme} vecteur propre, Mozilla, côté négatif

Nous avons choisi de fixer un seuil pour décider jusqu'à quel vecteur aller, pour constituer les communautés. Si la valeur propre est inférieure à 0.5% de la première valeur propre, elle est écartée. Cela donne typiquement une dizaine, voire une quinzaine de vecteurs propres, ce qui correspond visuellement à ce que l'on garderait « à la main ». Comme nous le vérifierons par la suite, ce facteur a peu d'importance.

Enfin, pour la dernière interrogation, concernant la quantité de sites à garder dans un vecteur propre donné, la réponse est similaire : le profil de courbe est à peu de choses près le même, voir figures 3 et 4 page suivante. Nous avons donc procédé de la même façon, en ne gardant que les sites dont la valeur d'authority est supérieure à 0.5% de la plus forte valeur d'authority.

a. L'algorithme

Étudions maintenant en détail l'algorithme que nous proposons, et qui permet de construire les communautés de HITS.

Celui-ci (algorithme 2 page suivante) a pour seule particularité de laisser chaque site dans la première communauté rencontrée (celle qui lui convient le mieux, puisque l'on commence par les vecteurs propres associés aux va-

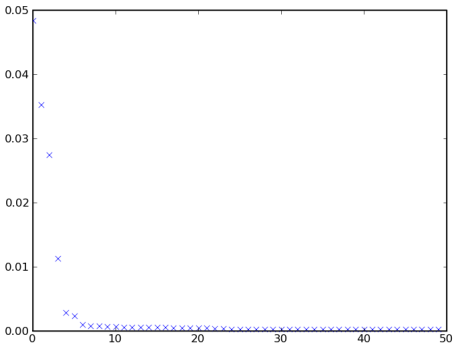

FIG. 4 – Décroissance des valeurs d'authority pour le 4ieme vecteur propre, Mozilla, côté négatif

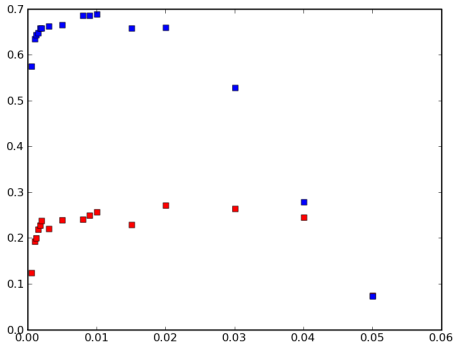

FIG. 5 – Pour différentes valeurs du facteur de seuil (en abscisse), de 0.5% à 5%, modularité obtenue par les communautés naïves (rouge, courbe inférieure) et par l'algorithme BGLL sur le même sous-graphe (bleu, courbe supérieure) – graphe de Mozilla

Données : Un graphe où chaque site est dans sa communauté, n sites au total
Les vecteurs propres v_i associés aux valeurs propres λ_i, triés par valeur propre décroissante
Les valeurs d'autorité a_{ij} et les sites correspondants s_{ij} contenus dans le vecteur v_i, triés par autorité croissante (ou décroissante, peu importe)
$i \leftarrow 1$;
tant que $\lambda_i > 0.05 \times \lambda_0$ **faire**
  $j \leftarrow 0$;
  tant que s_{ij} *a déjà été placé* **faire**
    $j \leftarrow j + 1$;
  fin
  $j_0 \leftarrow j$;
  tant que $a_{ij} > 0.05 \times a_{i0}$ **faire**
    si s_{ij} *n'a pas déjà été placé* **alors**
      mettre s_{ij} dans la communauté de s_{ij_0};
    fin
    $j \leftarrow j + 1$
  fin
  $j \leftarrow n$;
  tant que s_{ij} *a déjà été placé* **faire**
    $j \leftarrow j - 1$;
  fin
  $j_0 \leftarrow j$;
  tant que $a_{ij} > 0.05 \times a_{in}$ **faire**
    si s_{ij} *n'a pas déjà été placé* **alors**
      mettre s_{ij} dans la communauté de s_{ij_0};
    fin
    $j \leftarrow j - 1$
  fin
  $i \leftarrow i + 1$;
fin

Algorithme 2 : Algorithme de construction des communautés de HITS (version naïve)

leurs propres les plus grandes). Notons enfin que l'on commence au second vecteur car le premier vecteur n'a que des éléments de même signe.

Les deux parties similaires servent en fait à constituer d'abord la communauté du côté positif puis la communauté du côté négatif[15].

b. Résultats en terme de modularité

Toujours dans l'optique de répondre à la question « Les communautés de HITS sont-elles de bonnes communautés au sens de la modularité ? », nous allons mesurer la modularité des découpages en communautés donnés par l'algorithme 2.

Nous allons commencer par apporter une réponse au paramètre fixé arbitrairement à 0.5%. Regardons le graphe 5.

Il faut tout d'abord savoir que les points obtenus pour un facteur supérieur à 0.2 (soit 20%) ne sont pas significatifs, dans la mesure où le nombre de sites retenus est trop faible. On peut donner deux réponses :

- les valeurs de modularité ne bougent pas énormément, se stabilisant aux environs de 0.25, ce qui montre que **le facteur de seuil choisi n'est pas d'une importance cruciale** ;
- l'algorithme BGLL [BGLL08] donne toujours des résultats bien meilleurs : **les communautés de HITS données par l'algorithme 2 ont une modularité moyenne ; BGLL lancé sur le même sous-graphe exhibe un découpage en communautés donnant une bien meilleure modularité**.

Sur le graphe de l'IXXI, les résultats sont globalement identiques, à ceci près que l'on obtient des modularités meilleures en moyenne pour le découpage de l'algorithme 2 (environ 0.6)[16]. Néanmoins, là encore, l'algo-

[15] Notons bien que ces distinctions de négatif et de positif n'ont rien d'absolu puisque si x est vecteur propre, $-x$ l'est également.

[16] Ceci est dû au fait que le graphe de l'IXXI est plus petit (500 sites) et que HITS a tendance à exhiber, de manière récurrente dans les communautés, les sites institutionnels tels le CNRS, l'INRIA, Lyon-1, Lyon-2, l'ENS-Lyon, l'ENS-LSH. Ceux-ci ont naturellement de grosses boucles (liens d'eux vers eux-mêmes), ce qui renforce la modularité de manière « naturelle » : en effet, quel que soit le découpage en communautés, on est toujours assuré d'avoir beaucoup de liens au sein des communauté grâce à ces boucles, ce qui est précisément ce que mesure la modularité. Cette influence fera l'objet de la partie suivante.

rithme BGLL donne constamment de meilleurs résultats.

Le bilan, en termes de modularité, est donc relativement clair : les communautés de HITS créées par l'algorithme 2 donnent une certaine modularité, mais cela n'est pas comparable au regard de la modularité que peut exhiber un algorithme (BGLL) approprié pour le même graphe.

c. Étude qualitative des graphes obtenus

Malgré tout, le graphe des communautés de HITS, obtenu avec l'algorithme 2, possède une certaine modularité. À quoi ressemble-t-il concrètement ? Nous allons effectuer une étude qualitative sur deux exemples.

⋆ Mozilla

Sur l'exemple de Mozilla, une étude pratique du graphe créé par l'algorithme 2 page précédente permet de dégager les remarques suivantes :

- des sites pertinents à l'entourage de mozilla.org sont dégagés par l'algorithme : portail des développeurs Mozilla, Mozilla Europe, le magazine Mozilla, Add-ons de Firefox, Mozilla Corée, et ainsi de suite… dans la même communauté ;
- des sites importants sont égalements mis en évidence : Opera, Microsoft, Netscape, Del.icio.us ;
- beaucoup de sites « annexes » polluent le graphe, et ne constituent rien de très pertinent ;
- le graphe est constitué d'une grosse composante connexe exhibant une certaine structure communautaire – quelques sites isolés sont présents ;
- le découpage en communautés de HITS ne s'approche en aucun cas de la structure communautaire réelle du graphe, au sens de la modularité ;
- il n'y a réellement aucune valeur du paramètre de seuil qui donne des résultats corrects : trop élevé, et seuls trois ou quatre sites sont acceptés ; trop faible, et trop de sites sont acceptés.

Une particularité intéressante de ce processus est que, pour des valeurs de seuil « raisonnables » (1 ou 2%), le processus se comporte bien pour la plupart des vecteurs : seuls quelques sites sont conservés (de l'ordre de la dizaine). Cependant, pour quelques vecteurs (1 ou 2), le seuil est trop bas, et plus d'une centaines de sites sont acceptés. Ceci explique la présence de sites peu pertinents dans le graphe final. Élever la valeur de seuil ne résout pas la question, puisque pour les autres vecteurs, trop peu de sites sont conservés.

Naturellement, l'algorithme de détection de communautés par la modularité (BGLL) s'adapte parfaitement à la structure communautaire du graphe.

⋆ Ixxi

Les conclusions sont les mêmes sur le site de l'IXXI : des sites pertinents sont mis en évidence, comme les sites universitaires ENS-Lyon, les sites institutionnels comme l'INSA. Des sites annexes mais pertinents sont présents dans le graphe, comme des sites universitaires liés à l'IXXI. Les communautés formées par HITS sont de qualité correcte, dans la mesure où l'IXXI est dans la même communauté que l'ENS-Lyon, par exemple. Néanmoins, là encore, des sites annexes viennent polluer le graphe, et le découpage en communautés est loin d'être idéal.

d. Réflexion critique

De toute évidence, vouloir évaluer, de manière naïve, les communautés de HITS données par l'algorithme 2 à l'aide de la modularité est un échec. La modularité est faible, les communautés trouvées, de par le processus de seuillage, incluent parfois trop de sites, parfois pas assez. Le graphe résultant est par endroits pertinents, parfois totalement farfelu.

De plus, le processus utilisé nécessite une intervention humaine, pour choisir un seuil qui donne de bons résultats. Le seuil n'est pas le même selon les graphes, et dépend particulièrement de la taille du graphe posé en entrée.

De fait, et plutôt que de poursuivre des expérimentations heuristiques, en voulant à tout prix trouver des paramètres donnant de bons résultats (probablement sur un graphe en particulier, et pas sur les autres), nous avons préféré revoir le concept, en posant une question annexe.

> Est-il possible d'obtenir une bonne modularité à partir des communautés de HITS, et si oui, quel est le sens du graphe obtenu ?

3 Algorithme évolué (algorithme 3)

Nous avons alors naturellement cherché un meilleur algorithme qui permette de créer de « bonnes communautés » au sens de la modularité, tout en gardant les critères d'appartenance à une communauté selon HITS. L'idée est ici de faire intervenir de manière plus importante la modularité, afin que l'on puisse d'une part se garder des erreurs grossières (cent sites retenus pour un vecteur) et d'autre part obtenir une modularité correcte, ce qui sera gage d'un graphe pertinent à la fois en termes sémantiques (grâce à HITS) et en termes de modularité.

La modularité va être la fonction que nous chercherons à optimiser, et nous verrons que grâce à ce critère, nous pourrons obtenir un découpage en communautés qui suit les communautés de HITS, mais qui possède en même temps une bonne modularité.

a. Algorithme

L'algorithme 3 page suivante procède comme suit. On commence par sélectionner les premiers vecteurs propres pour lesquels la valeur propre associée est suffisamment élevée – c'est à dire que la communauté a une importance significative selon HITS[17]. Notons k leur nombre. On a ensuite des positions (variables posp et posn) dans chacun des vecteurs propres, du côté positif (posp) et du côté négatif (posn). Rappelons que les éléments de ces vecteurs sont triés, et précisons que s_{ij} est le j^{ieme} site du i^{ieme} vecteur propre.

On maintient tout au long de l'algorithme l'ensemble des nœuds constituant le sous-graphe construit. Cette variable nœuds est initialement une liste vide.

[17] Opération effectuée à la main pour l'instant, le temps a manqué pour définir un critère efficace permettant de choisir automatiquement combien de vecteurs propres conserver. On peut cependant penser à un seuillage comme dans la partie précédente.

Données :
- Un graphe G où chaque site est dans sa communauté, n sites au total
- Les k premiers vecteurs propres v_i associés aux valeurs propres λ_i, triés par valeur propre décroissante
- Les valeurs d'autorité a_{ij} et les sites correspondants s_{ij} contenus dans le vecteur v_i, triés par autorité croissante (ou décroissante, peu importe)
- Un nombre N représentant le nombre d'itérations de l'algorithme

posp $\leftarrow [0, \cdots, 0]$;
posn $\leftarrow [n, \cdots, n]$;
nœuds $\leftarrow [\,]$;
pour *i allant de* 1 *à* N **faire**
  $m^* \leftarrow 0$;
  `// meilleure modularité trouvée`
  op $\leftarrow$ (None,None);
  pour *j allant de* 1 *à* k **faire**
    $sg \leftarrow$ `SousGraphe` (G, nœuds $\cup s_{i,\text{posp}[j]}$);
    `// pour le graphe sg qui est temporaire`
    mettre $s_{i,\text{posp}[j]}$ dans la communauté de $s_{i,0}$;
    $m \leftarrow$ `Modularité` (sg);
    si $m > m^*$ **alors**
      $m^* \leftarrow m$;
      op $\leftarrow$ (j, "+");
    fin
    $sg \leftarrow$ `SousGraphe` (G, nœuds $\cup s_{i,\text{posn}[j]}$);
    `// pour le graphe sg qui est temporaire`
    mettre $s_{i,\text{posn}[j]}$ dans la communauté de $s_{i,n}$;
    $m \leftarrow$ `Modularité` (sg);
    si $m > m^*$ **alors**
      $m^* \leftarrow m$;
      op $\leftarrow$ (j, "-");
    fin
  fin
  si $m^* > 0$ **alors**
    $j \leftarrow$ op[0];
    si *op[1]* = "+" **alors**
      nœuds $\leftarrow$ nœuds + $s_{i,\text{posp}[j]}$;
      `// pour le graphe G`
      mettre $s_{i,\text{posp}[j]}$ dans la communauté de $s_{i,0}$;
      posp[j] $\leftarrow$ posp[j] + 1;
    fin
    si *op[1]* = "-" **alors**
      nœuds $\leftarrow$ nœuds + $s_{i,\text{posn}[j]}$;
      `// pour le graphe G`
      mettre $s_{i,\text{posn}[j]}$ dans la communauté de $s_{i,n}$;
      posn[j] $\leftarrow$ posn[j] - 1;
    fin
  fin
fin
Résultat : `SousGraphe` (G, nœuds)

Algorithme 3 : Algorithme de construction des communautés de HITS (version évoluée)

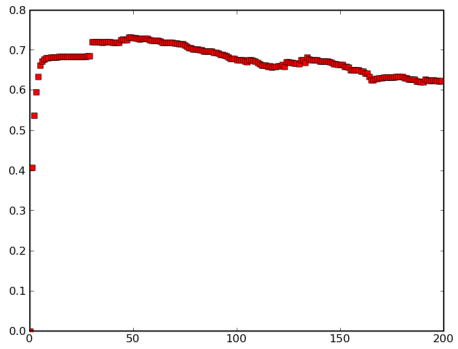

FIG. 6 – Évolution de la modularité, Mozilla, $k = 15$, 200 itérations, avec boucles

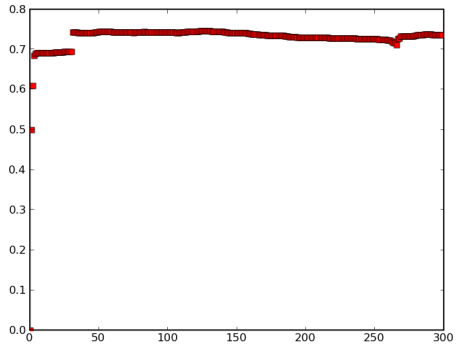

FIG. 7 – Évolution de la modularité, Suez, $k = 10$, 300 itérations, avec boucles

À chaque itération (variable i), on parcourt l'ensemble des vecteurs propres (variable j), et l'on regarde la modularité obtenue en ajoutant le posp[j]$^{\text{ieme}}$ site du j^{ieme} vecteur propre au sous-graphe, en le plaçant dans la communauté du premier site du j^{ieme} vecteur. On regarde également la modularité obtenue en mettant le posn[j]$^{\text{ieme}}$ site du j^{ieme} vecteur propre au sous-graphe, dans la communauté du n^{ieme} site du j^{ieme} vecteur. À l'issue de l'itération, c'est l'opération (variable op) donnant la meilleure modularité qui est effectuée. La modularité est la meilleure sur toutes les opérations possibles, mais elle n'est pas nécessairement meilleure que la modularité du sous-graphe courant.

Les vérifications techniques permettant de se garder de situations improbables ont été épargnées au lecteur[18].

[18] Afin de dissiper tout malentendu, donnons un exemple. Dans l'hypothèse où l'on exécute l'algorithme un nombre très grand d'itérations, il peut arriver que la position posp dans un vecteur propre soit celle du dernier site possédent une valeur d'authority positive, parce que tous les autres ont été placés précédemment (un vecteur propre peut avoir beaucoup plus d'éléments positifs que négatifs). On ne peut donc pas placer le site situé au-delà de cette position car c'est un site appartenant à une autre communauté.

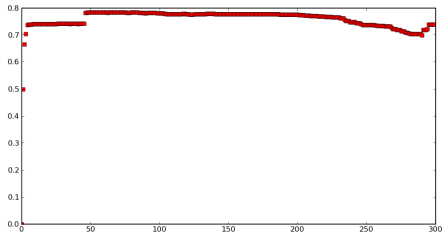

FIG. 8 – Évolution de la modularité, Suez, $k = 15$, 300 itérations, avec boucles

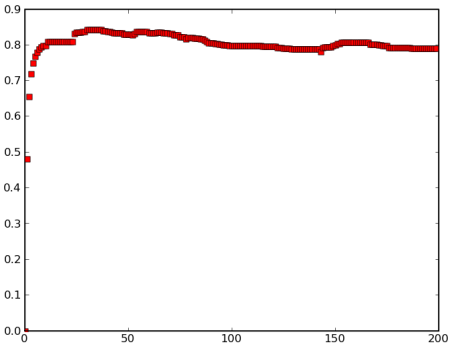

FIG. 9 – Évolution de la modularité, Ixxi, $k = 15$, 200 itérations, avec boucles

b. Résultats en terme de modularité

Les graphiques 6 page précédente, 7 page ci-contre, 8 et 9 montrent l'évolution de la modularité lors de l'exécution de l'algorithme 3, et ce pour différents sites, avec différents paramètres.

On pourra tout d'abord remarquer que de très fortes modularités sont atteintes : jamais moins de 0.7, souvent 0.85. Ce sont des valeurs élevées mettant en évidence une forte structure communautaire. Si l'on compare ces modularités avec les performances de l'algorithme BGLL sur le sous-graphe donnant la modularité maximale, les résultats sont sensiblement équivalents. Parfois même BGLL n'arrive pas à atteindre la modularité du sous-graphe découpé en communautés selon HITS. Il est intéressant de remarquer également que la modularité décroît, en règle générale, lentement, une fois le maximum atteint. C'est-à-dire que l'on garde une bonne modularité au fur et à mesure que l'on ajoute des nœuds. La structure communautaire reste forte lors de l'ajout de nœuds.

La plupart des opérations consistent à déplacer des nœuds de communauté en communauté : en effet, un nœud se retrouve souvent dans les premières positions de plusieurs vecteurs propres. Il sera donc amené à changer de communauté, ce qui permet d'avancer la position dans le vecteur propre pour, à terme, rencontrer de nouveaux sites. Au final, après 200 itérations, seuls 91 nœuds ont été ajoutés pour Suez. Lorsqu'on observe une remontée de la modularité, c'est le plus souvent dû à l'arrivée de nouveaux nœuds dans le sous-graphe.

c. Étude qualitative des graphes obtenus

Nous étudions ici les sous-graphes obtenu au moment où la modularité est maximale (sauf mention contraire). Il en sera de même dans la partie suivante, lorsque nous effectuerons une nouvelle étude qualitative.

⋆ Suez

Dans le cas de Suez, on a un graphe exhibant une assez forte structure communautaire : on repère deux grands ensembles distincts :

- Suez et les sites liés à l'environnement : Suez, Suez International, Suez Environnement, Energy Assistance, Nations Unies, Association Lesseps… ;
- Sites liés à la production d'énergie : Areva, Cadarache, Saclay, European Synchotron, Ministère de la défense, Polytechnique, traitement des déchets nucléaires, CEA… ;

Le graphe obtenu est en lui-même plus clair, dans la mesure où l'on peut distinguer à l'œil nu différentes communautés (ensembles de nœuds densément liés) : on ne retrouve pas le fouillis de nœuds sans rapport apparent que donnait l'algorithme précédent.

Au niveau du découpage en communautés, les deux ensembles ci-dessus sont placées dans des communautés différentes par l'algorithme 3. De même, l'algorithme met (judicieusement) dans une troisième communauté des sites « gouvernementaux » comme le Centre national de la fonction publique, les Assedic, le ministère de l'éducation, Paris Diderot…

Le graphe n'est cependant pas exempt de bruit : parmi les sites sans intérêt qui ont été retenus, on notera le site du créateur du site web de Suez, un blog qui, parce qu'il mentionne des dizaines d'entreprises dont Suez, a acquis un certain poids, SourceForge, EuroNext, Adobe…Ces sites ne sont pas directement pertinents car l'on s'attend plutôt à voir, dans l'entourage de Suez, des sites directement liés à l'eau, l'environnement, des sites liés à la réputation de Suez sur la toile, le positionnement par rapport à ses concurrents. Mais ces sites font partie de manière intrinsèque de l'environnement de Suez, et il faut en tenir compte : la structure de l'environnement d'un site web n'est pas aussi claire que ce que l'on pourrait penser initialement.

L'algorithme BGLL se comporte de manière « standard » sur le graphe : il isole bien les communautés selon la structure topologique du graphe, ce qui favorise les sous-sujets lorsqu'ils sont topologiquement reliés, ce qui est le cas ici.

Au final, l'algorithme HITS (algorithme 3) donne une modularité de 0.7726 ; l'algorithme BGLL appliqué au même graphe obtient 0.8023, soit moins de 4% d'amélioration.

⋆ Ixxi

On remarque deux pics dans la courbe de la modularité. Le graphe obtenu lors du premier pic contient peu de sites, mais ce sont des sites essentiels : ainsi, l'algorithme met dans une même communauté le site de l'ENS Lyon, de l'INRIAlpes, de `http://perso.ens-lyon.fr`, de l'IMAG. L'algorithme garde également le site de LSH, et

de l'INSA, ainsi que d'autres sites universitaires importants.

Le second pic donne un graphe qui présente également une structure communautaire, mais plus d'une centaine de sites sont retenus. Il n'est donc pas très intéressant de s'attarder sur son étude.

Pour ce qui est du graphe obtenu au moment du premier pic, quelques sites peuvent être vus comme du bruit, mais on remarque que les sites les plus « centraux » ont été conservés. Ceci est dû au fort nombre de liens internes : les fortes boucles que possèdent des sites comme le CNRS les favorisent. Néanmoins, dans ce cadre précis, ceci est bénéfique : les sites mineurs n'ont pas beaucoup de liens internes et cèdent la place aux sites institutionnels. Le graphe donné est donc, globalement, de bonne qualité.

Le découpage en communautés par BGLL donne une modularité de 0.8402 contre 0.8435 pour le découpage en communautés par HITS à l'aide de l'algorithme 3. C'est l'un des très rares cas où BGLL n'arrive pas à donner une meilleure modularité que HITS. Le graphe est cependant très petit (une dizaine de nœuds) et le découpage en communautés des deux algorithmes semble pertinent.

4 Analyse critique des deux algorithmes

Si l'on compare cet algorithme évolué (algorithme 3) avec l'algorithme naïf (algorithme 2 page 10), il y a un progrès indéniable : il n'y a plus d'arrivée soudaine d'une centaine de sites. Même s'il y a un bruit qu'il n'est pas possible de passer sous silence, les sites retenus semblent nettement plus pertinents que dans la version naïve.

Néanmoins, la présence d'un fort bruit dans tous ces graphes conduit à formuler quelques interrogations. Pourquoi des sites a priori peu pertinents par rapport au sujet principal sont-ils sélectionnés ? Est-ce un hasard que la modularité donnée par HITS soit aussi forte ?

Il apparaît en fait que la présence des boucles sur les *sites* influence très fortement les résultats. Par exemple, si lors de la collecte de données, on arrive sur un blog, de nombreux liens vont être explorés : des liens d'archives, de catégories, de billets précédents, les liens du menu de navigation, les liens de commentaires…et chaque nouvelle page contiendra elle-même des liens internes à foison : dans l'exemple de Suez, la présence d'un blog dans le sous-graphe s'explique par une boucle portant un poids de 7000…alors que la plupart des arêtes portent des poids n'excédant pas 100, et se situant plus fréquemment aux alentours de 10.

L'influence sur la modularité est donc non-négligeable : en effet, quel que soit la communauté dans laquelle le site est placé, le poids des liens intra-communautaires sera toujours très fort. **Le fort poids des boucles rend n'importe quel découpage en communautés satisfaisant.**

Pour obtenir des résultats plus pertinents, nous avons choisi de supprimer les boucles, pour ne plus biaiser les résultats de HITS et des calculs de modularité.

PARTIE V
NOUVELLE ÉTUDE SANS LES BOUCLES

L'étude a donc été refaite sans les boucles présentes sur les sites. Le processus suivi sera le même que dans la partie précédente. Nous passerons cependant rapidement sur l'algorithme naïf (algorithme 2 page 10) pour nous focaliser sur l'algorithme évolué (algorithme 3 page 12), car il donne des résultats plus fructueux. Nous verrons que le fait d'enlever les boucles permet une analyse plus fine.

Enlever les boucles implique la présence d'une nouvelle matrice pour HITS puisque la matrice d'adjacence du graphe a été modifiée. Néanmoins, le profil de décroissance des valeurs propres et des valeurs d'authority dans un vecteur propre reste le même, ce qui justifie que l'on lance l'algorithme naïf (algorithme 2) de nouveau.

1 Avec l'algorithme naïf (algorithme 2)

L'influence des boucles est ici très clairement mise en évidence. Avec les boucles, on obtenait des modularités de 0.25, qui sont certes faibles, mais laissent supposer qu'il existe une structure communautaire du graphe [19]. Ceci était dû, comme nous l'avons expliqué, au poids des boucles qui augmentent considérablement la proportion de liens intra-communautaires, augmentant ainsi la modularité même pour un mauvais découpage en communautés. Ici, cette « aide » ne s'applique plus, on trouve donc des modularités inférieures à zéro, proches de -0.2. Le découpage n'a plus rien de communautaire, et se rapproche d'un découpage au hasard.

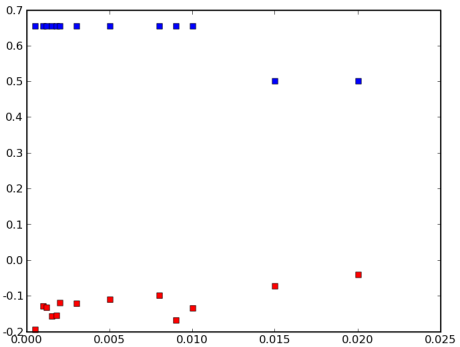

FIG. 10 – Modularité donnée par l'algorithme naïf sans les boucles (en rouge, courbe inférieure) ; modularité donnée par BGLL sur le même graphe (bleu, courbe supérieure) – Ixxi.fr

Le fait de supprimer les boucles permet d'apporter dans ce cas très précis une réponse plus claire, et plus tranchée que lorsque nous avions les boucles : maintenant que les données biaisant l'algorithme ont été supprimées, il est évident que le découpage en communautés de

[19] On considère que l'on a exhibé une structure communautaire lorsque la modularité dépasse 0.3.

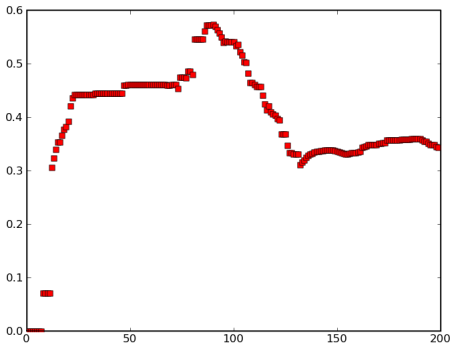

FIG. 11 – Modularité donnée par 200 itérations de l'algorithme évolué, sans les boucles, 15 vecteurs propres – Mozilla.org

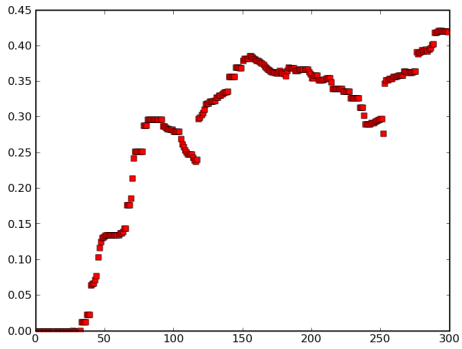

FIG. 12 – Évolution de la modularité, Suez, $k = 15$, 300 itérations, sans boucles

HITS donné par l'algorithme 2 n'a strictement rien à voir avec les communautés au sens de la modularité.

Ce résultat n'a, somme toute, rien de surprenant. L'algorithme HITS poursuivant un but tout à fait différent des algorithmes de détection de communauté, et même s'il existe des méthodes de détection de communautés à base de valeurs propres, le rapport entre les deux semble inexistant.

2 Avec l'algorithme évolué (algorithme 3)

Tout n'est pas perdu cependant car on peut espérer obtenir quand même des modularités correctes avec l'algorithme 3 page 12.

a. Évolution de la modularité

⋆ Mozilla

L'exemple de la figure 11 est extrêmement intéressant, il est à mettre en perspective avec la figure 6 page 12. La courbe n'est tout d'abord plus aussi lisse, et stagne quelques temps à zéro : en effet, les premiers ajouts de nœuds ne donnent pas un graphe connexe, mais des ensembles disjoints de nœuds, ne possédant aucune arête puisque les boucles ont été évacuées. La modularité est donc nulle. Au bout de quelques itérations, l'algorithme commence à produire des sous-graphes intéressants, et l'on retrouve un pic beaucoup plus prononcé que dans la figure 6 page 12. En revanche, la modularité obtenue est plus faible.

BGLL, sur le graphe obtenu au moment du pic, trouve une modularité de 0.7511 contre 0.5727 pour les communautés de HITS. L'écart est plus fort que pour la version avec boucles, et on ne voit en aucun cas HITS donner de meilleures communautés que BGLL sans les boucles.

Le sous-graphe possède 49 nœuds (environ 500 au total dans le graphe de l'IXXI) ce qui confirme le comportement observé précédemment : on a une meilleure modularité lorsque l'on a créé un sous-graphe des sites les plus « importants ».

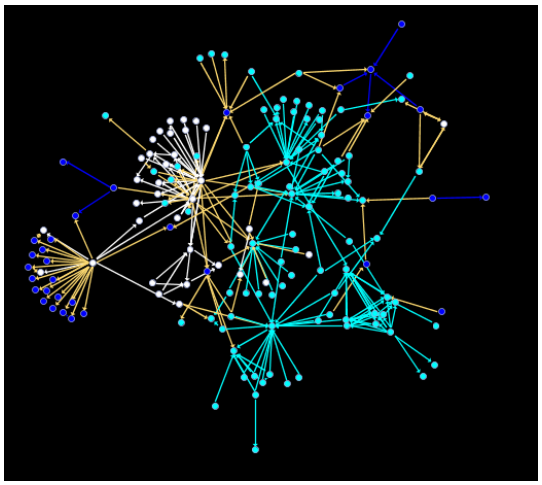

FIG. 13 – Découpage en communautés de HITS sur le graphe obtenu au pic de modularité (Suez). Graphique original en couleurs, disponible au format PDF en ligne.

⋆ Suez

Comme on le voit à la figure 12, dans le cas de `http://www.suez.fr`, on obtient plusieurs pics. Les phases de décroissance sont probablement dues au déplacement de nœuds, tandis que les phases de croissance correspondent à l'arrivée de nouveaux nœuds.

La modularité obtenue est plus forte que celle obtenue pour Mozilla.org, et le graphe qui en résulte possède effectivement une structure communautaire.

b. Étude qualitative des graphes obtenus

De manière générale, le processus (algorithme 3) basé sur HITS donne un graphe dont les communautés sont de très grande taille. Sur un graphe de 182 nœuds (Suez), l'algorithme 3 ne garde que 3 communautés, là où BGLL en fera une douzaine. Les deux processus diffèrent donc fondamentalement, et ce de manière plus marquée que dans la partie précédente, dans leur découpage.

Le bruit est cependant moindre : par exemple, dans l'environnement de Suez, on ne retrouve pas de blogs comme c'était le cas auparavant. L'algorithme se focalise

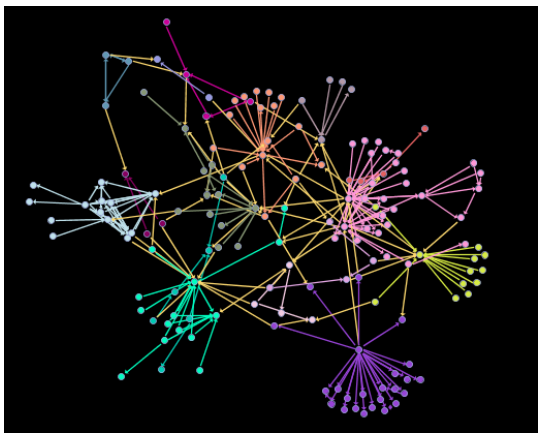

FIG. 14 – Découpage en communautés de BGLL sur le graphe obtenu au pic de modularité (Suez). Graphique original en couleurs, disponible au format PDF en ligne.

plus sur les sites institutionnels qui reçoivent beaucoup de liens, par exemple.

★ Suez

Le processus donne pour le sous-graphe obtenu après quelques centaines d'itérations un excellent aperçu de l'environnement de Suez.fr : le graphe présente effectivement une structure communautaire marquée. La topologie est relativement claire, et le graphe est lisible.

En étudiant de près le découpage en sites, il en ressort que l'algorithme 3, comme nous l'avons dit, ne favorise pas un découpage en communautés très fin. Cependant, il fait preuve de quelques subtilités : par exemple, il sépare de sites d'entreprise les sites du CNFPT (fonction fublique), du Ministère de l'éducation, et de Paris Diderot. BGLL n'est pas capable de telles subtilités, car ces nœuds sont proches topologiquement de Adobe.fr par exemple : l'algorithme aboutit à un regroupement de tous ces nœuds dans une même communauté.

On retrouvera les graphes correspondant à ces deux illustrations aux figures 14 et 13 page précédente.

★ Mozilla

Le résultat est, de même que pour Suez, très satisfaisant pour Mozilla. L'algorithme 3 arrive à dégager différents ensembles de sites dans l'entourage de Mozilla. Les sites choisis sont très pertinents, et l'on voit clairement que des communautés ont été mises en évidence : tous les sites ayant trait à Mozilla (portail des développeurs, projet Seamonkey, Addons, Mozillazine, Getfirefox, Bugzilla) sont regroupés. Des sites appartenant à un même domaine (`???.example.com`) sont tout naturellement regroupés dans une même communauté. Enfin, là encore, l'algorithme 3 page 12 fait preuve de petites subtilités que BGLL, sur le même graphe, n'est pas capable d'effectuer.

PARTIE VI
CONCLUSION, PERSPECTIVES

On pourra trouver un récapitulatif des différents algorithmes dans le tableau 1 page 18.

1 Une vue d'ensemble

L'idée initiale était d'extraire, de l'environnement d'un site web, des communautés de sites dont la pertinence est assurée grâce à HITS, puis de voir si ces communautés étaient valides au sens de la modularité.

Une première tentative a consisté à prendre, naïvement, les sites les plus pertinents retournés par HITS dans chaque communauté, en constituer un sous-graphe, et mesurer la modularité dudit sous-graphe. Ceci a été concrétisé dans l'algorithme 2. Les résultats ont été décevants, ce qui nous a amenés à mettre au point une autre approche : ne pas prendre « les premiers sites de chaque communauté » mais plutôt remplir progressivement les communautés de HITS en maximisant la modularité à chaque étape : c'est l'algorithme 3. Cette nouvelle façon de procéder devait, a priori, favoriser HITS : en rajoutant la contrainte d'une bonne modularité, on était en droit d'attendre à la fois un bon découpage selon HITS, et un bon découpage en termes de modularité.

Cette approche a semblé, dans un premiers temps, favorable : en réalité, la présence de boucles sur les sites faussait totalement les résultats, et même si certains sites institutionnels, de par leur grand nombre de liens internes, étaient mis en avant, de nombreux sites secondaires, tels des blogs, étaient faussement favorisés.

Nous avons donc refait une étude sans les boucles sur les graphes, ce qui a confirmé les performances toutes à fait mauvaises de l'algorithme naïf (algorithme 2) en termes de modularité.

2 Que nous a apporté la combinaison des deux algorithmes ?

La partie la plus féconde concerne les comparaisons entre les algorithmes de découpage en communautés HITS et BGLL (voir section 2 page précédente), une fois les boucles enlevées. Deux choses importantes ont été mises en évidence.

Tout d'abord, HITS n'est pas particulièrement adapté à la détection de communautés : les communautés mises en évidence par HITS ne s'adaptent que faiblement à la topologie du graphe. HITS constitue de larges communautés, certes s'accordant à la topologie du graphe, mais place par exemple dans une même communauté des nœuds qui ne sont pas reliés... ce qui est plutôt déconcertant. Les communautés sont trop grandes, et on distingue aisément à l'œil nu des sous-communautés qui auraient gagné à être séparées. Ainsi, chercher à maximiser les résultats de HITS en termes de modularité tend à donner, au final, un découpage de faible qualité, qui ne reflète pas les qualités de HITS.

Ensuite, la détection de communautés ne se prête pas nécessairement au Web. Il n'est pas trivial de considé-

rer, parce que des nœuds sont dans une même zone topologique, que sémantiquement, ils sont à rapprocher. Un blog fera par exemple des liens vers des sujets différents : faut-il mettre tous ces sites dans une même communauté ? De plus, on a vu que là où HITS arrivait néanmoins à montrer ses qualités, la détection de communautés passait à côté de détails *qu'il était possible de remarquer*, à la fois pour un humain regardant le graphe, et pour un algorithme.

Au final, on en vient à s'interroger sur la légitimité de la modularité. Remarquablement adaptée dans le cas de réseaux sociaux, **graphes non orientés, sans poids sur les arêtes**, on comprend moins bien sa signification sur les graphes orientés. La plupart des articles se contentent d'indiquer que leur méthode est facilement adaptable aux graphes orientés – ce qui est vrai. Mais l'exemple du graphe du web montre qu'on est loin de comprendre précisément le concept de communautés pour ce type de graphes.

Cependant, et c'est là l'apport de cette méthode, la combinaison de HITS et de la modularité permet de sélectionner efficacement les sites les plus pertinents dans les vecteurs propres non principaux. Les articles mettant en évidence la dissociation des sous-sujets de HITS dans les vecteurs propres non principaux se contentaient d'effectuer une étude manuelle, en regardant pour chacun des premiers vecteurs propres non principaux quels étaient les sites possédant les plus fortes valeurs d'authority. Ici, on arrive à effectuer une sélection de manière algorithmique, et l'on se garde des déboires rencontrés en utilisant l'algorithme naïf (arrivée subite de plus de cent sites dans le sous-graphe).

3 Perspectives

L'algorithme évolué (algorithme 3) gagnerait probablement à être amélioré. Peut-être est-il possible d'éviter les nombreuses opérations de déplacement de nœuds. Peut-être peut-on encore améliorer les performances de HITS en termes de modularité. Mais le plus important semble de mener un travail théorique visant à mieux comprendre le concept de communautés dans des graphes pondérés et orientés. Beaucoup de travail reste à fournir en amont : que faire des graphes dynamiques comme le graphe du Web ? Comment s'adapter au fait que l'on ne pourra toujours que voir une *fraction infime* du graphe du Web ? Faut-il nécessairement analyser le contenu textuel des pages afin de comprendre précisément la structure ?

Il est clair grâce à cette étude que la modularité souffre de défauts indéniables. Néanmoins, la détection de communautés reste un champ de recherche extrêmement prometteur et des avancées rapides sont à attendre dans ce domaine.

	Avec les boucles	Sans les boucles
Algorithme 2 (naïf)	Du fait des énormes boucles, on arrive à avoir des modularités qui sont correctes, mais c'est illusoire. Ce n'est dû qu'à la présence quasi-systématique de boucles sur chaque nœud, ce qui donne quel que soit le découpage en communautés, même le plus anti-modulaire possible, une modularité qui semble bonne. BGLL arrive sans surprise à extraire une modularité meilleure.	Si l'on enlève les boucles, le découpage en communautés donne des modularités négatives, plus aucune structure communautaire n'est créée. BGLL arrive cependant toujours à extraire des modularités « bonnes » de ces sous-graphes (0.5 à 0.6).
Algorithme 3 (évolué)	Comme il y a toujours de grosses boucles, la courbe est très régulière mais fortement influencée par le poids des boucles. On arrive cependant à des résultats pertinents comme sur le graphe de l'IXXI (on a le CNRS, l'INRIA, l'INRIAlpes, l'ENS, Lyon1 dans la même communauté). BGLL n'arrive qu'à faire un tout petit peu mieux que la modularité que donne l'algo HITS, voire moins bien. Mais de toute façon la modularité n'est pas très intéressante puisque totalement biaisée par le poids des boucles. La modularité reste très forte, au-delà de 0.8, voire jusqu'à 0.9.	Cette partie est la plus parlante. On obtient de bons résultats (voir graphe Suez) où l'on voit que HITS conserve sa valeur sémantique en regroupant certains sites institutionnels. BGLL lui regroupe les nœuds en fonction de la topologie du graphe, ce qui est moins pertinent dans le cas des sites web. Ainsi, BGLL a une meilleure modularité que celle donnée par HITS (0.2 de mieux en général). La modularité reste dans l'ensemble *moyenne* (0.5, 0.6).

TAB. 1 – Tableau récapitulatif

PARTIE VII
BIBLIOGRAPHIE

Références

[AK08] Gregoire Allaire and Sidi Mahmoud Kaber. *Numerical Linear Algebra*. Springer, 2008.

[BGLL08] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of community hierarchies in large networks. Preprint, 2008.

[CDG+99] Soume Chakrabarti, Byron E. Dom, David Gibson, Jon Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Mining the link structure of the world wide web. 1999.

[CNM04] Aaron Clauset, M. E. J. Newman, and Christopher Moore. Finding community structure in very large networks. *Physical Review E.*, 70(6), 2004.

[FB06] Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. In *Proceedings of the National Academy of Sciences of the United States*, 2006.

[GKR98] David Gibson, Jon Kleinberg, and Prabhakar Raghavan. Inferring web communities from link topology. In *HYPERTEXT '98: Proceedings of the ninth ACM conference on Hypertext and hypermedia : links, objects, time and space—structure in hypermedia systems*, pages 225–234, New York, NY, USA, 1998. ACM.

[KL01] Jon Kleinberg and Steve Lawrence. The structure of the web. *Science*, 294:1849–1850, 2001.

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Journal of the ACM*, volume 46, 1999.

[KRRT99] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the web for emerging cyber-communities. In *WWW '99: Proceedings of the eighth international conference on World Wide Web*, pages 1481–1493, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[LWP06] Feng Luo, James Z. Wang, and Eric Promislow. Exploring local community structures in large networks. In *WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 233–239, Washington, DC, USA, 2006. IEEE Computer Society.

[New04] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69, 2004.

[New06] M. E. J. Newman. Modularity and community structure in networks. In *PNAS*, 2006.

[NG04] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2), 2004.

[PFP+07] Gergely Palla, Illes J. Farkas, Peter Pollner, Imre Derenyi, and Tamas Vicsek. Directed network modules. *New journal of physics*, 9(6):186, 2007.

[PL05] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *Lecture notes in computer science*, 2005.

[WT07] Ken Wakita and Toshiyuki Tsurumi. Finding community structure in mega-scale social networks: [extended abstract]. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1275–1276, New York, NY, USA, 2007. ACM.

PARTIE VIII

FIGURES ET ALGORITHMES

Table des figures

Liste des Algorithmes