Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Grau en Matemàtiques

Treball de fi de grau

Comunitats en xarxes complexes: detecció i centralitat

Alexi Quintana Mathé

Director: Francesc Comellas, Departament de Matemàtiques

Juny 2017

Agraeixo a en Francesc la seva ajuda

Abstract

Community structure is a recurrent feature of graphs representing real data, known also as complex networks. This means their nodes are organized in clusters or communities, that are, loosely speaking, parts of the graph with more edges inside each cluster than edges among different clusters. This work is divided in two sections: first we compare four different community detection algorithms, and in the second we study the relationship between the centrality of nodes and their position in their community. Six different centrality measures have been considered, and three different types of random graphs with known community structure have been generated for statistical purposes.

Paraules clau

Xarxes complexes, comunitats, centralitat, algoritmes de detecció de comunitats.

Índex

1. Introducció i preliminars

1.1 Estructura del treball i objectius

En aquest treball estudiem l'estructura de comunitats en xarxes complexes. S'anomenen xarxes complexes als grafs construïts a partir de dades empíriques complint una sèrie de propietats, i s'entén per comunitat en un xarxa complexa, en línies generals, a una part del graf on la densitat de connexions és més gran que a la resta del graf. Als apartats 1.2 i 1.3 es detallen amb més precisió aquests conceptes.

El punt de vista emprat ha estat eminentment pràctic, i es poden dividir les investigacions realitzades en dues seccions clarament diferenciades. Per una banda, a la secció 2 del treball s'ha comparat la precisió de quatre algoritmes de detecció de comunitats, l'algoritme Girvan-Newman, el Louvain, el Miracom, i l'Infomap. La manca d'una definició rigorosa de comunitat, i la complexitat de la qüestió, han donat peu en els últims vint anys a una multitud d'algoritmes diferents enfocant el problema des de punts de vista molt diversos [11]. Esquemàticament, el procediment ha consistit en la creació de grafs aleatoris amb una estructura de comunitats donada, als quals se'ls han aplicat els algoritmes per avaluar fins quin punt han identificat les comunitats correctament. L'objectiu d'aquest part del treball és doncs determinar quins algoritmes distingeixen millor les comunitats, i quines característiques de la xarxa a tractar afecten a la seva precisió

Per una altra banda, en la secció 3 del treball s'ha volgut traçar una relació entre l'estructura de comunitats d'una xarxa i la centralitat dels seus nodes. Existeixen diferents mesures de centralitat en una xarxa, destinades a esbrinar quins són els seus nodes més importants en base a diferents criteris. Més concretament, en aquesta secció s'han calculat correlacions entre la proporció de connexions internes i externes a la comunitat d'un node, i la seva centralitat. S'han considerat sis mesures de centralitat diferents. L'objectiu principal és doncs determinar si en una xarxa complexa aquesta correlació és generalment significativa, és a dir, si la situació d'un node a la seva comunitat afecta a la seva centralitat respecte el graf en el seu conjunt. A més, també es vol esbrinar quins són els factors que defineixen el signe de la correlació, i la seva intensitat, sobretot en relació a les característiques de l'estructura en comunitats de la xarxa, i en funció del tipus de centralitat. Per a això s'han fet servir els grafs aleatoris del paràgraf anterior, i una sèrie de xarxes construïdes a partir de dades empíriques.

Per al compliment d'aquest objectius s'ha programat en Python, fent servir el paquet NetworkX, que inclou un conjunt complet d'estructures de dades i càlculs ja implementats per a treballar fàcilment amb grafs. La representació gràfica d'algunes xarxes a la secció 3 s'ha realitzat fent servir el programari Gephi. Al llarg de tot el treball s'ha tractat només amb grafs o xarxes no dirigits i sense pesos a les arestes, per a no complicar excessivament l'estudi.

En els apartats següents d'aquesta secció s'introdueixen els conceptes de xarxa complexa i de comunitats en un graf, introduïnt definicions que es faran servir al llarg del treball. S'expliquen també les sis mesures de centralitat d'un node que es fan servir a la secció 3, i el procés de construcció i les característiques dels grafs aleatoris amb estructura de comunitats que s'han utilitzat per fer proves a les seccions 2 i 3.

1.2 Xarxes complexes

El concepte de xarxa es troba present a l'imaginari col·lectiu de gran part de la població. En la seva versió més simple es tracta senzillament de nodes, o elements d'un conjunt qualsevol, connectats entre ells en funció d'un criteri determinat. Així doncs, a través de les xarxes, representades matemàticament mitjançant

grafs, es poden modelitzar una multitud de sistemes diferents, focalitzant l'anàlisi en les relacions entre els elements que els conformen més que en les seves propietats individuals. Alguns exemples serien la xarxa 'World Wide Web', on els nodes són pàgines web, que es connecten asimètricament entre elles si una té un hiperenllaç a l'altre; xarxes neuronals, on els nodes són neurones i les arestes representen sinapsis; xarxes socials, on els nodes poden ser persones i les connexions s'estableixen en base a alguna pregunta empírica respecte les seves relacions, o xarxes d'aeroports, que es connecten si existeix algun vol directe de una a l'altre [24].

Tots aquests exemples són el que s'anomenen xarxes complexes, degut a una sèrie de característiques compartides per a totes elles, i que les allunyen dels grafs purament aleatoris, o generats amb regularitats clares, que es tracten normalment en Teoria de Grafs. La complexitat s'entén, en un sistema en general, com emergent de la interacció dels diferents elements d'aquest sistema, sense cap coordinació centralitzada d'aquests [19]. Tot i ser un concepte molt difícil de definir, la idea central gira al voltant de l'existència d'un grau d'impredictibilitat en el comportament dels agents, que no és totalment regular, ni totalment aleatori. Alguns d'aquests patrons comuns a moltes xarxes complexes, tot i que aquestes puguin provenir d'àmbits tant diversos com els mostrats, són els següents [16]:

- El 'small-world efffect', segons el qual la distància mitjana entre dos nodes de la xarxa segueix una escala logarítmica respecte el nombre de nodes d'aquesta. Així, fins i tot en xarxes complexes immenses la distància entre dos nodes tendeix a ser petita.
- Un coeficient de clústering elevat. El coeficient de clústering d'un graf és la proporció mitja de veïns d'un node que també estan connectats entre ells.
- Presència de cliques, és a dir, conjunts de nodes amb totes les connexions possibles entre ells, i de comunitats, conjunts més amplis de nodes on la densitat de connexions entre ells és major que la mitjana del graf.
- Correlació entre les característiques d'un node i les dels seus veïns, denominada normalment assortativitat quan els nodes de més grau es troben connectats entre ells.
- 'Scale free graphs': es refereix a la característica d'alguns grafs, en els quals la distribució de graus dels seus nodes segueix aproximadament una llei de potències. Això causa l'aparició dels denominats 'hubs', és a dir, nodes amb un grau molt més elevat que la mitjana.

A partir de la teoria de grafs, desenvolupada quasi exclusivament des de les matemàtiques, ha sorgit l'estudi de les xarxes construïdes amb dades reals, és a dir, adoptant un punt de vista destinat a desenvolupar eines que permetin descriure i analitzar l'estructura darrere d'aquestes xarxes empíriques. Es presenten a continuació, en els punts 1.3 i 1.4, la descripció d'alguns d'aquests conceptes, no estudiats en teoria de grafs, i que s'han fet servir però en el desenvolupament d'aquest treball.

1.3 El problema de les comunitats en xarxes complexes

Tal i com s'ha dit, la presència de comunitats o clústers (al llarg del treball es faran servir ambdós termes indistintament) és una característica particular de les xarxes complexes. Tot i això, no existeix una definició rigorosa del concepte de comunitat, i és inevitable un cert grau d'arbitrarietat i l'aparició d'ambigüitats alhora de determinar les comunitats d'un graf [11].

Si $C \subset V$ és un subgraf del graf $G(V, E)$, es defineixen el grau intern $\delta_C^{in}(v_i)$ i extern $\delta_C^{out}(v_i)$ de $v_i \in V$, respecte el subgraf C, com el nombre d'arestes existents entre v_i i altres nodes de C, i el nombre d'arestes entre v_i i nodes de $V \backslash C$, respectivament. Quan la comunitat o subgraf de pertinença estigui especificada,

es denotarà simplement $\delta^{in}(v_i)$ i $\delta^{out}(v_i)$. A partir d'aquí es poden definir, ampliant les definicions habituals de densitat d'arestes en un graf, la densitat interna i externa d'arestes per a C, $\delta^{in}(C)$, i $\delta^{out}(C)$, on n_C és la quantitat de nodes a C:

$$\delta^{in}(C) = \frac{\sum_i \delta_C^{in}(v_i)}{n_C(n_C-1)/2}$$

$$\delta^{out}(C) = \frac{\sum_i \delta_C^{out}(v_i)}{n_C(n-n_C)}$$

Per a que C sigui una comunitat, cal que $\delta^{in}(C)$ sigui clarament més gran que la densitat total del graf $\delta(G) = 2m/n(n-1)$, si m és el nombre d'arestes del graf i n el de nodes. Aquest, però, és només un criteri general, i n'hi ha diversos més que, a partir d'aquí, defineixen el concepte de comunitat d'una manera o altra [11]. Una observació important que es deriva d'aquestes definicions és que només pot haver-hi estructura de comunitats en una xarxa si la quantitat d'arestes és de l'ordre de la quantitat de nodes, és a dir, en grafs relativament poc densos.

En particular, una eina que es fa servir habitualment és comparar el graf en qüestió amb un graf aleatori sense estructura de comunitats que compleixi algunes de les seves propietats estructurals. Fent servir aquesta eina es pot definir un criteri de qualitat, és a dir, una mesura que avaluí com de 'bona' és una partició en comunitats del graf. Tot i que, com s'ha dit, no existeix un consens, i probablement sigui impossible, sobre què és una 'bona' partició en comunitats, si que es pot escollir una definició determinada que permeti escollir un cert criteri de qualitat. Un dels criteris de qualitat més populars és l'anomenada modularitat d'una partició, que s'explica a continuació.

Modularitat

Sigui G un graf, i $\mathbf{A}$ la seva matriu d'adjacència. Denotem per $\mathbf{P}_{ij}$ el nombre esperat d'arestes entre els vèrtexs i i j, en el graf aleatori amb el que es vol efectuar la comparació. Aleshores, si $\delta(C_i, C_j)$ és una funció que pren valor 1 quan els vèrtexs i i j estan a la mateixa comunitat, i 0 quan no ho estan, es pot escriure la modularitat com [22]

$$Q = \frac{1}{2m}\sum_{ij}(\mathbf{A}_{ij} - \mathbf{P}_{ij})(\delta(C_i, C_j))$$

Veiem doncs com la modularitat, definida així, considerarà més adequada a la partició en comunitats que tingui, en mitjana, més arestes internes a cada clúster respecte les que tindria el graf aleatori considerat.

A l'hora d'escollir el model de graf amb el qual comparar la xarxa que s'està analitzant, es podria escollir un graf amb el mateix nombre de nodes i arestes que l'original, i amb aquestes col·locades amb la probabilitat mitjana d'una aresta al graf original. Aquesta opció generaria però un graf amb una distribució gaussiana per als graus, la qual cosa no es compleix a la realitat, i per això s'introdueix una modificació: que la distribució de graus esperada per al graf aleatori sigui la mateixa que a l'original. Si en un graf aleatori k_i i k_j són els graus dels nodes i i j, la probabilitat que s'estableixi una aresta entre ells és $p_{ij} = k_i k_j/4m^2$, ja que $p_i = k_i/2m$ i $p_j = k_j/2m$ són les probabilitat de que s'esculli aleatòriament una 'mitja aresta' sortint de k_i i de k_j, respectivament, d'un total de $2m$ mitges arestes. Es considera doncs que es forma una aresta entre i i j si coincideixen les dues mitges arestes, és a dir, amb probabilitat $p_i p_j$. El nombre esperat de connexions entre els vèrtexs i i j serà doncs $\mathbf{P}_{ij} = 2mp_i p_j = k_i k_j/2m$, arribant a l'expressió final de la modularitat:

$$Q = \frac{1}{2m}\sum_{ij}(\mathbf{A}_{ij} - \frac{k_i k_j}{2m})(\delta(C_i, C_j))$$

Amb aquesta definició, la modularitat pren valors entre -1 i 1, amb el 0 corresponent, entre d'altres possibilitats, a la partició on tot el graf és una mateixa comunitat. Si cap partició dona valors més grans que 0, s'interpreta que la xarxa no té estructura de comunitat [11]. Cal tenir en compte que la modularitat augmenta amb el nombre de nodes del graf, per la qual cosa no és una bona mesura comparativa si les xarxes tenen mides molt diferents [15]. Un altre dels seus errors és l'anomenat límit resolutiu de la modularitat, que dificulta la detecció de comunitats amb una mida més petita que un cert factor depenent de la mida total de la xarxa [12]. Tot i això, és un criteri de qualitat molt utilitzat per els algoritmes de detecció de comunitats, i s'ha generat molta literatura científica al seu voltant. A la secció 2 es descriuen els diferents algoritmes escollits per a l'anàlisi comparativa, i així doncs el criteri que fa servir cadascun per determinar les comunitats.

1.4 Mesures de centralitat

La noció de centralitat d'un node juga un paper molt important en l'anàlisi de les xarxes complexes. La distribució dels graus seguint una llei de potències, i la presència d'estructures com les comunitats causa que alguns nodes siguin més importants que d'altres, i les mesures de centralitat permeten identificar aquests nodes en base a diferents criteris. A continuació es presenta una explicació de les diferents mesures de centralitat que es faran servir a la secció 3 del treball. Es poden separar en dos grups: les que es basen en els fluxos, o, en altres paraules, en les transferències d'alguna quantitat entre els nodes de la xarxa, des d'un punt de vista dinàmic, podríem dir, i les que fan servir l'estructura topològica del graf, i assignen centralitat a un node donant més importància al seu entorn més proper. Les quatre primeres centralitats presentades formen part del primer grup, i les dues últimes del segon. És important tenir en compte que la centralitat dels nodes d'un graf n'és un invariant, és a dir, que no depèn del seu etiquetatge.

Centralitat de proximitat

La centralitat de proximitat és una de les mesures de centralitat conceptualment i analíticament més senzilles. Així, simplement considera més centrals els nodes que en mitjana es troben més a prop, en termes de nombre de vèrtexs en el camí geodèsic més curt entre dos vèrtexs, de la resta de nodes del graf. Si $l_i = \frac{1}{n-1}\sum_{j\neq i} d_{ij}$, on d_{ij} és la distància entre dos nodes, és la distància mitja entre el node i i els altres nodes del graf, aleshores:

Definició 1.1. Sigui G un graf connex no dirigit i sense pesos a les arestes. La centralitat de proximitat C_i del node i és [13]

$$P_i = \frac{1}{l_i} = \frac{n-1}{\sum_{j\neq i} d_{ij}}$$

És útil tenir en compte que, degut a que les distàncies entre nodes en una xarxa complexa tendeixen a ser petites, la centralitat de proximitat pren habitualment valors força similars entre uns nodes i els altres [24]. És per això que pot resultar una mesura poc útil per discriminar efectivament entre nodes més i menys centrals.

Centralitat d'intermediació

Aquesta mesura també fa servir els camins més curts entre els nodes d'un graf per a calcular-ne la centralitat. En aquest cas, però, no és la distància el que importa, sinó el nombre de camins més curts que passen

per un node donat. Així, un node serà més central com més d'aquests camins passin per ell, ja que la informació, o la quantitat que sigui fluint per la xarxa tendirà a passar per a aquest node. També seran els nodes amb una centralitat d'intermediació ("betweenness centrality"en anglès) més alta els que, si s'extreuen de la xarxa, en dificultaran més la comunicació.

Definició 1.2. Sigui G un graf connex, no dirigit, i sense pesos, $\sigma(s,t)$ el nombre total de camins més curts entre els nodes s i t, i $\sigma(s,t|v)$ el nombre de camins més curts entre s i t que passen pel node v. A més, si $s=t$, $\sigma(s,t)=1$, i si $v \in s,t$, $\sigma(s,t|v)=0$. Aleshores la centralitat d'intermediació de v és [5]

$$I(v)=\sum_{s,t\in V}\frac{\sigma(s,t|v)}{\sigma(s,t)}$$

Veiem com en aquesta definició es té en compte el cas en que hi han diversos camins més curts entre dos nodes, i aleshores es divideix l'increment en centralitat que aporta cada un a v per el nombre que n'hi hagi. En la implementació en aquest treball de la centralitat d'intermediació s'ha normalitzat per $2/((n-1)(n-2))$, ja que no es conta que un camí de s a t passi per aquests mateixos vèrtexs.

Una crítica freqüent a les mesures de centralitat presentades és que només tenen en compte els camins més curts en un graf [24]. En molts casos, però, els fluxos en una xarxa no passen només per aquests camins més curts, i és per això que s'han proposat mesures alternatives. Partint de la base d'una xarxa elèctrica, modelitzada com un graf amb pesos positius a les arestes, que indiquen la seva conductivitat, Brandes i Fleischer [6] defineixen la centralitat de proximitat en flux ('current flow closeness centrality') i la centralitat d'intermediació de flux ('current flow betweenness centrality'). La idea és que considerar unes quantitats de corrent que entren (font) o surten (pou) de determinats nodes del graf, de forma a que la suma total d'aquestes quantitats sigui 0, determina un conjunt de funcions possibles que assignen a cada aresta el corrent que hi passa, fent sortir de cada font, i arribar a cada pou, la quantitat de corrent corresponent. D'aquest conjunt de funcions possibles, les lleis de Kirchoff en determinen una única, que serà el denominat corrent elèctric de la xarxa. Si $G(V,E)$ és el graf, $N(G,c)$ és la xarxa elèctrica amb $c: E \longrightarrow \mathbb{R}_{\geq 0}$ els pesos de cada aresta, i si $\overrightarrow{E}$ és el conjunt d'arestes considerant les dues orientacions possibles per a cada una, aleshores $x: \overrightarrow{E} \longrightarrow \mathbb{R}_{\geq 0}$ denota el corrent elèctric. En el cas que ens interessa, es considerarà que la funció de subministrament $b: V \longrightarrow \mathbb{R}$ que assigna a cada node el corrent que hi entra o surt, externament a la xarxa, senzillament introdueix una unitat de corrent per un vèrtex s, i la fa sortir per un altre t. S'anomenarà doncs b_{st}, i x_{st} al corrent elèctric corresponent. Amb aquests preliminars es pot passar a definir ambdues mesures alternatives de centralitat, en el cas de grafs sense pesos a les arestes, que són els tractats en aquest treball.

Centralitat d'intermediació en flux

A partir d'una funció de subministrament b_{st}, i del corrent x_{st} que determina segons les lleis de Kirchoff, es pot definir el flux de corrent travessant un vèrtex $v \in V$, $v \neq s,t$:

$$\tau_{st}(v)=\frac{1}{2}\Big(\sum_{\substack{e\in E\\ v\in e}}|x(\overrightarrow{e})|\Big)$$

Aleshores:

Definició 1.3. Sigui $N = (G, c)$ una xarxa elèctrica. La centralitat d'intermediació en flux $IF : V \longrightarrow \mathbb{R}_{\geq 0}$ es defineix com [6]

$$IF(v) = \frac{1}{(n-1)(n-2)} \sum_{\substack{s,t \in V \\ s,t \neq v}} \tau_{st}(v)$$

Així doncs, amb aquesta mesura de centralitat s'aplica una idea similar a la centralitat d'intermediació, al valorar la centralitat d'un node en funció dels camins entre dos altres nodes els quals es troba, però en aquest cas no es tenen en compte només els més curts, sinó tots, donant més o menys pes a uns o altres en funció del corrent elèctric que hi passaria si fos una xarxa elèctrica. Com que passa més corrent com més curt és el camí entre dos nodes, sembla una mesura raonable. Tot i això, pot resultar estrany modelitzar la centralitat en una xarxa social, per exemple, aplicant el que passa en una xarxa elèctrica. Es pot demostrar però que aquesta mesura és equivalent, numèricament com a mínim, a la denominada centralitat de camins aleatoris [23] , que si que té sentit substantiu per a una xarxa complexa genèrica. Aquesta mesura compta el nombre de vegades, en mitjana, que un passeig aleatori entre cadascun dels parells de vèrtexs del graf passa per un node donat per assignar-li una centralitat determinada. Si el camí hi passa en un sentit, i després en l'altre, s'anul·la però la seva aportació, per evitar que si un passeig aleatori es limita a anar i tornar d'un node a un dels seus veïns, sense dirigir-se enlloc, s'incrementi la centralitat d'aquests dos nodes.

Es poden comparar la centralitat d'intermediació estàndard, i la centralitat d'intermediació en flux, situant-les a un extrem i l'altre d'un espectre possible de camins a tenir en compte a l'hora de decidir com de central és un node en funció de per a quants d'aquests camins actua d'intermediador [24]. Un extrem seria contar només els camins més curts, suposant que els fluxos en la xarxa fan servir només aquests, i l'altre extrem seria contar-los com si els fluxos es dirigissin aleatòriament en el graf. Depenent del cas, la realitat es trobarà doncs probablement entre un d'aquests dos extrems, i sovint resulta adient tenir en compte l'interval creat per ambdues centralitats alhora de determinar la centralitat efectiva d'un node.

Centralitat de proximitat en flux

Per a poder definir aquesta mesura es necessita primer introduir una nova magnitud present en les xarxes elèctriques: el voltatge o diferència de potencial $\hat{p}(\overrightarrow{e}) : \overrightarrow{E} \longrightarrow \mathbb{R}$, definit per la llei d'Ohm de forma a que $\hat{p} = x(\overrightarrow{e})/c(e)\ \forall e \in E$. Aleshores, es diu que un vector $p : V \longrightarrow \mathbb{R}$ assigna potencials absoluts si $\hat{p}(v, w) = p(v) - p(w)\ \forall (v, w) \in \overrightarrow{E}$. Igual que abans, denotem $\hat{p}_{st}$ el potencial absolut d'una xarxa elèctrica quan només hi arriba una unitat de corrent per el node s, i aquest surt pel node t.

Definició 1.4. Sigui $N = (G, c)$ una xarxa elèctrica. La centralitat de proximitat en flux $PF : V \longrightarrow \mathbb{R}_{>0}$ es defineix com [6]

$$PF(s) = \frac{n-1}{\sum_{t \neq s} \hat{p}_{st}(s) - \hat{p}_{st}(t)} \quad \forall s \in V$$

En el cas d'aquest treball s'ha tractat només amb grafs sense pesos a les arestes, i per tant $\hat{p}(\overrightarrow{e}) = x(\overrightarrow{e})$. Es pot veure com, en aquestes condicions, el que mesura la centralitat de proximitat en flux d'un node és la mitjana de la distància d'aquest node a la resta, però tenint en compte per a calcular aquesta distància no només al camí més curt, sinó tots els camins. Cada camí "afegeix"proximitat al node, i es ponderen per la seva llargada, de manera a que els més curts realitzen una aportació més gran. Aquesta interpretació sorgeix més directament de l'equivalència entre la centralitat de proximitat en flux tal i com s'ha definit, i la denominada centralitat informativa ('Information centrality') [6]. Aquesta mesura,

desenvolupada per Stephenson i Zelen [27], es defineix introduint la noció d'un 'camí combinat' entre un node i un altre, format per una suma ponderada amb uns certs pesos de tots els camins possibles entre aquests dos nodes. Els pesos s'escullen com la inversa del nombre d'arestes que conté el camí, que es denomina la seva informació. Així, la informació del camí combinat és la suma d'aquests pesos per a tots els camins possibles entre els dos nodes. Si I_{st} és la informació del camí combinat entre els vèrtexs s i t, es defineix la centralitat informativa $I(s)$ del vèrtex s com

$$I(s) = \frac{n}{\sum_{t \in V} 1/I_{st}},$$

si es determina que $I_{ss} = \infty$. Es pot establir doncs una analogia similar a la comentada per la centralitat de flux d'intermediació respecte la centralitat d'intermediació, ja que amb la centralitat de proximitat en flux o informativa també es considera que tots els camins entre dos nodes incrementen la proximitat entre ells, d'alguna manera, i no només els camins més curts. En aquest cas, a més, es veu clarament com modificant els pesos a la definició de la informació d'un camí combinat es pot decidir fàcilment a quins dels camins entre dos nodes volem donar més o menys importància.

Centralitat de vector propi

El grau d'un node és la mesura de centralitat més senzilla que podem utilitzar, de forma a que els nodes de grau més alt son senzillament els més centrals. Podem pensar, però, que no tots els veïns d'un node són igual d'importants a l'hora de mesurar la seva centralitat. Així doncs, la idea darrere de la centralitat de vector propi ("eigenvector centrality"en anglès) és que confereixen més centralitat a un node les connexions amb nodes que, alhora, són també força centrals. La centralitat de vector propi d'un node serà proporcional a la suma d'aquesta mesura per als seus veïns. [24]

Formalment, inicialitzem el vector $\mathbf{x}$, on volem que x_i sigui la centralitat de vector propi del vèrtex i, suposant que $x_i = 1 \ \forall i$. Si $\mathbf{A}$ és la matriu d'adjacència del graf, l'equació $\mathbf{x}(1) = \mathbf{A}\mathbf{x}$ imposa que els valors de $\mathbf{x}(1)$ siguin combinació lineal de les centralitats de vector propi inicials representades en $\mathbf{x}$, i $\mathbf{A}$ determina els elements de la combinació lineal, que són els veïns del node. A partir d'aquí definim el procés iteratiu $\mathbf{x}(t) = \mathbf{A}^t\mathbf{x}(0)$. Quan $t \to \infty$ obtindrem doncs la centralitat de vector propi de cada node. Podem però, escrivint $\mathbf{x}(0)$ com una combinació lineal dels vectors propis $\mathbf{v}_i$ de $\mathbf{A}$, $\mathbf{x}(0) = \sum_i c_i\mathbf{v}_i$, simplificar aquest procés:

$$\mathbf{x}(t) = \mathbf{A}^t \sum_i c_i\mathbf{v}_i = \sum_i c_i k_i^t \mathbf{v}_i = k_1^t \sum_i c_i (\frac{k_i}{k_1})^t \mathbf{v}_i \Longrightarrow \underset{t\to\infty}{\mathbf{x}(t)} = c_1 k_1^t \mathbf{v}_1$$

, amb k_1 el valor propi més gran de $\mathbf{A}$. Veiem com, d'aquesta manera, el vector $\mathbf{x}$ de centralitat de valor propi és simplement proporcional al vector propi del valor propi més gran de la matriu d'adjacència del graf, per la qual cosa compleix $\mathbf{A}\mathbf{x} = k_1\mathbf{x}$. De tots els múltiples del vector propi possibles, prenem senzillament el mateix vector propi de valor propi més gran. El paquet NetworkX que s'ha fet servir utilitza el mètode de les potències per trobar-lo.

Centralitat de subgraf

L'última mesura de centralitat utilitzada en aquest treball suposa també, en part, una millora a la centralitat bàsica que ens ofereix el grau, i alhora fa servir també els camins existents en una xarxa. Desenvolupada per Estrada i Rodríguez-Velázquez [10], la centralitat de subgraf o de 'comunicabilitat' ('communicability centrality'), mesura la importància d'un node en funció del nombre de camins sortint i arribant a aquest

node, ponderant l'aportació de cada camí en funció de la seva llargada. La quantitat de camins d'una llargada donada k que surten i arriben a un node i d'un graf ve donada per $(\mathbf{A}^k)_{ii}$, si $\mathbf{A}$ és la matriu d'adjacència del graf. Així, la centralitat de subgraf seria quelcom similar a

$$\sum_{k=0}^{\infty} \frac{(\mathbf{A}^k)_{ii}}{c_k}$$

, essent c_k unes constants determinades. Per a que la sèrie convergeixi, però, cal que aquests c_k creixin prou ràpid. S'escull doncs $c_k = k!$, de manera a que es defineix la centralitat de subgraf com

$$\sum_{k=0}^{\infty} \frac{(\mathbf{A}^k)_{ii}}{k!}.$$

Així doncs, si λ és el valor propi més gran de $\mathbf{A}$, $\sum_{k=0}^{\infty} \frac{(\mathbf{A}^k)_{ii}}{k!} \leq \sum_{k=0}^{\infty} \frac{\lambda^k}{k!} = e^{\lambda}$, i la sèrie convergeix. Es poden però escollir coeficients c_k diferents, donant peu a mesures diferents. Amb $c_k = k!$, el grau del node correspon al terme $k = 1$ de la sèrie, i queda clar doncs com la centralitat de subgraf l'incorpora com una part important de la mesura.

Es denomina centralitat de subgraf a aquesta quantitat perquè depèn directament dels diferents subgrafs, de mida petita, en els quals es trobi el node en qüestió [9]. En altres paraules, el nombre de camins de longitud 1,2, 3, 4, o 5, que són els que mes pes tenen en el càlcul, depèn de si el node es troba situat en molts o pocs subgrafs de 2, 3 o 4 nodes, és a dir, si té un grau alt però també si els seus veïns i nodes propers tenen ells també forces connexions entre ells. En certa manera, és una mesura que ens indica si l'entorn proper al node està molt densament connectat.

1.5 Grafs aleatoris utilitzats

Tant per a fer-los servir a la secció 2 com a la secció 3 del treball, s'han generat diferents tipus de grafs aleatoris amb estructura de comunitats. L'objectiu és que, disposant de molts grafs de cada tipus, es puguin calcular estadístiques, de la precisió a l'hora de detectar comunitats dels diferents algoritmes seleccionats, i de la correlació entre la posició d'un node a la seva comunitat i la seva centralitat. A continuació s'expliquen les característiques de cadascun d'aquests grafs, i com s'han generat.

Grafs amb comunitats estàndard

Els grafs aleatoris amb comunitats més senzills a l'hora de fer tests són els generats amb l'anomenat "planted l-partition" model. Aquest algoritme genera un graf amb l comunitats, un nombre de nodes g per a cadascuna, i unes probabilitats p_{in} d'aresta intra-comunitat i p_{out} d'aresta inter-comunitat. Cada comunitat es construeix com un sub-graf aleatori del tipus Erdös-Rényi amb probabilitat de connexió entre dos nodes qualsevols p_{in} i, per a cada parella possible de nodes pertanyents a clústers diferents, es genera una aresta entre ells amb probabilitat p_{out} [11]. En aquest treball s'ha fet servir la generalització d'aquest model implementada a Networkx, que permet crear comunitats de la dimensió que es desitgi. Anomenarem als grafs aleatoris generats d'aquesta manera grafs amb comunitats estàndard.

Per a afegir diversitat i comprovar com aquesta produeix diferències en el funcionament dels algoritmes de detecció de comunitats i en la correlació entre la proporció d'arestes internes i externes a la comunitats de cada node i la seva centralitat al graf, s'han escollit tres parells de probabilitats (p_{in}, p_{out}) diferents: $(0.6, 0.08), (0.25, 0.08)$, i $(0.25, 0.1)$. A més, per a cadascun d'aquests parells s'han creat grafs amb dos

estructures de comunitats diferents, representades com un vector amb la quantitat de nodes de cada comunitat a cada posició: $(32, 32, 32, 32)$ i $(16, 16, 32, 64)$. Es tenen doncs sis tipus de grafs amb comunitats estàndard, amb dues combinacions per a (p_{in}, p_{out}), la primera i la tercera, corresponents a grafs amb una estructura de comunitats clarament definida, i una altra, la segona, on no és tant clara. A més, en cada cas variarà força el grau mig dels nodes. S'han creat 25 grafs de cadascun dels sis tipus, imposant que siguin connexos per a evitar així problemes amb els algoritmes de detecció de comunitats i el càlcul de les diferents mesures de centralitat.

LFR benchmark

Com s'ha vist, els nodes dels grafs aleatoris que s'acaben de descriure tindran un grau força similar, dins d'una mateixa comunitat. Com ja s'ha dit, una característica habitual de les xarxes que representen sistemes reals és que presenten nodes amb graus molt diferents entre ells, i la distribució dels seus graus segueix, aproximadament, una llei de potències. A més, sembla que les dimensions de les comunitats també segueixen una llei de potències, la qual cosa no es compleix per als grafs amb comunitats estàndard. Així doncs, cal una altra aproximació a la creació de grafs aleatoris amb comunitats per a fer proves que puguin ser transportables a xarxes construïdes amb dades reals de forma més fiable.

Lancichinetti, Fortunato, i Radicchi [21] ofereixen un algoritme que crea grafs aleatoris, on el grau dels nodes, i la dimensió de les comunitats, segueixen ambdós una distribució de potències. El procediment de creació és, esquemàticament, el següent: S'escull el nombre de nodes N del graf, i se li assigna a cada node un grau extret d'una llei de potències amb exponent γ. El grau mig k, i γ, venen donats, i es talla la distribució de potències a uns graus mínim i màxims k_{min} i k_{max} en funció de k. A continuació es seleccionen les mides de les comunitats a partir d'una llei de potències amb exponent β, de manera a que la suma de totes doni N, tallant també la distribució en un mínim i un màxim $c_{min} > k_{min}$ i $c_{max} > k_{max}$. S'assigna a cada node una comunitat aleatòriament, i a partir d'un paràmetre μ donat, es connecten una fracció $1 - \mu$ del seu grau amb altres vèrtexs de la seva comunitat, i una fracció μ amb vèrtexs d'altres comunitats, també aleatòriament. Al llarg d'aquest procés es produeixen diverses re-assignacions dels nodes a una comunitat, i no es compleix que μ sigui exactament el ràtio entre δ^{out} i $\delta^{in} + \delta^{out}$ per a cada node, però si s'arriba a una bona aproximació. Anomenarem als graf aleatoris creats d'aquesta manera grafs LFR.

Per a aquest treball s'ha fet servir el codi ofert per el propi Fortunato a la seva web: <`https://sites.google.com/site/santofortunato/inthepress2`>. Els paràmetres escollits han estat $N = 128$, $k = 13$, $k_{max} = 25$, $c_{min} = 8$, $\gamma = -2$, $\beta = -1$, i s'han fet servir dos valors per a μ diferents: $0,2$ i $0,5$. S'han creat 25 grafs per a cadascun d'aquests valors de μ. Els paràmetres han estat seleccionats de manera a que les comunitats no fossin excessivament petites, i es pugui veure la diferència entre els grafs que es generen amb $\mu = 0,2$, és a dir, comunitats ben definides, i $\mu = 0,5$, és a dir, comunitats poc definides. Finalment, cal tenir en compte que la manera en que estan construïts els grafs LFR, a partir d'un paràmetre μ que és aproximadament el mateix per a cada node, i no de probabilitats de connexió interna i externa a una comunitat, com en el cas dels grafs amb comunitats estàndard, evita la possibilitat de que alguns nodes tinguin més connexions amb una altra comunitat que amb la que teòricament se'ls ha assignat.

Grafs regulars amb comunitats

A més d'aquests dos tipus de grafs aleatoris, que són els més freqüentment utilitzats a l'hora de realitzar proves amb algoritmes de detecció de comunitats [21], s'ha decidit emprar un tercer tipus de graf: grafs aleatoris regulars amb estructura de comunitats. L'objectiu, més que de trasllat dels resultats obtinguts

a xarxes reals, és analític, i sobretot enfocat a la secció 3 del treball. Així doncs, amb aquests grafs es podrà veure l'efecte real que té sobre la centralitat d'un node el fet que connecti entre una comunitat i una altra, és a dir, que tingui una gran proporció d'arestes fora de la seva comunitat, sense tenir en compte el seu grau. Els grafs creats tenen quatre comunitats de 32 nodes cada una, i se n'han generat de dos tipus diferents, en funció de la proporció d'arestes entre comunitats que tenen. Els anomenarem grafs regulars amb comunitats, i el grau dels seus nodes és sempre 15. L'algoritme usat per a generar-los és el següent:

1. Es comença creant quatre grafs aleatoris regulars de 32 nodes cada un i grau 15, a partir de la comanda que ofereix el paquet Networkx, basada en l'article de Steger i Wormald [26].
2. A continuació es desconnecta un vèrtex aleatori de cadascun d'aquests grafs amb un dels seus veïns, escollit també aleatòriament. Si ja no té veïns a la comunitat, es selecciona un altre.
3. Ara s'uneixen els 8 nodes desconnectats aleatòriament entre ells, però sempre els d'una comunitat amb els d'una altra. Així es pot mantenir el compte de la quantitat d'arestes entre cada comunitat i la resta del graf.
4. Es torna al punt dos, tantes vegades com es desitgi en funció de com de ben definides es vol que estiguin les comunitats.

Veiem com a cada iteració dels passos 2 i 3, es treu una aresta interna a cada comunitat, i es creen dues arestes entre cadascuna i la resta del graf. Per al primer tipus de graf es realitza aquest procés 16 vegades en total, per la qual cosa, de les $15 \cdot 16$ arestes internes a cada comunitat, en queden $14 \cdot 16$, és a dir, 14 connexions internes de mitja per vèrtex. Alhora, es generen 32 connexions entre una comunitat i la resta del graf, 1 de mitja per a cada vèrtex. Podem traduir aquestes dades a les probabilitats p_{in} i p_{out} dels grafs amb comunitats estàndards: de les $(31 \cdot 32)/2$ arestes internes possibles, en tenim $(14 \cdot 32)/2$, per la qual cosa $p_{in} = 14/31 \cong 0,45$, i, de manera similar, $p_{out} = 32/(32 \cdot 96) = 1/96 \cong 0,010$. Pel que fa a la correspondència amb els grafs LFR, senzillament tindríem, per a aquest tipus de grafs, $\mu = 1/15$. L'algoritme per al segon tipus de graf s'ha iterat pels passos 2 i 3 un total de 112 vegades, i amb càlculs similars veiem com això resulta en 8 connexions internes a la comunitat i 7 externes de mitja per vèrtex. També es pot veure com això correspon a $p_{in} = 8/31 \cong 0,25$ i $p_{out} = 7/96 = 0,07$, i a $\mu = 7/15$, i per tant les comunitats en aquests grafs estan aproximadament tant clarament definides com la segona combinació de (p_{in}, p_{out}) en els grafs amb comunitats estàndard, i els LFR grafs amb $\mu = 1/2$.

Per a tenir la relació entre els paràmetres que determinen com de ben definides estan les comunitats en els grafs aleatoris en cada cas, entre tots els considerats, només falta calcular a quina μ correspondrien les probabilitats $(p_{in}, p_{out}) = (0.6, 0.08)$ i $(p_{in}, p_{out}) = (0.25, 0.01)$ dels grafs amb comunitats estàndard. Cal tenir en compte que les diferències en l'estructura de comunitats escollida per a aquests grafs causen diferències en aquesta relació, i que en el paràgraf anterior s'estava tractant implícitament amb l'estructura (32,32,32,32). Així doncs, s'arriba a que per $(p_{in}, p_{out}) = (0.6, 0.08)$, amb aquestes comunitats la correspondència seria $\mu = 0.29$, i amb l'estructura (16,16,32,64) seria $\mu = 0,26$. Quan $(p_{in}, p_{out}) = (0.25, 0.01)$, en canvi, en el primer cas $\mu = 0.12$, i en el segon $\mu = 0.10$. En el cas de comunitats diferents, la μ corresponent varia substancialment a cada comunitat, i s'ha calculat doncs la mitja ponderada per a tots els nodes.

Cal tenir en compte però les diferències en els processos de generació de cada tipus de graf, que fan que les correspondències que s'han calculat no siguin més que il·lustratives. Així, per als grafs regulars amb comunitats, la mitjana d'arestes internes i externes calculada per a cada node és exacta, mentre que en el cas dels graf LFR és aproximada, i és també quasi el mateix valor que el de cada node. Finalment, en el

cas dels grafs amb comunitats estàndard es tracta només d'un càlcul que s'ha realitzat, però l'aleatorietat aresta per aresta en la que es basa el seu algoritme generatiu fa que es pugui allunyar força dels valors calculats. En relació amb això, i com ja s'ha dit, el fet de que pels grafs LFR μ siguin un valor fix per a tots els nodes fa que, mentre sigui igual o menor a 0,5, la pertinença a la comunitat assignada teòricament durant l'algoritme estigui assegurada. En canvi, pels altres dos tipus de graf aleatoris es pot donar el cas, sobretot pels que tenen comunitats poc definides, que un node assignat a una comunitat tingui de fet més connexions amb una altra.

2. Detecció de comunitats

2.1 Procediment

Com ja s'ha dit, l'objectiu d'aquesta part del treball és efectuar una anàlisi comparativa de la precisió de quatre algoritmes de detecció de comunitats diferents, l'algoritme Girvan-Newman, el Louvain, el MiraCom, i l'Infomap. Més endavant s'explica el seu funcionament, en línies generals. La idea és veure, a partir dels diferents grafs aleatoris amb una estructura de comunitats coneguda, fins quin punt s'hi ajusta l'estructura de comunitats trobada per cadascun dels algoritmes.

A l'hora de determinar el grau amb que s'ajusten les comunitats trobades amb les conegudes a priori es poden fer servir diferents mesures [11]. Es poden trobar algunes basades en la quantitat de parells de vèrtexs classificats en la mateixa o en diferents comunitats en cada cas, en la coincidència dels clústers, és a dir, buscant quins parells de clústers de cada classificació corresponen millor, i, finalment, en teoria de la informació. En aquest treball s'ha fet servir una mesura del tercer tipus, la informació mútua normalitzada, de les més usades per a la realització de tests amb algoritmes de detecció de comunitats [20] [11].

Informació mútua normalitzada

La connexió entre teoria de la informació i la comparació de diferents estructures de comunitats parteix de la idea que, si dos particions dels vèrtexs d'un graf són similars, es necessita molt poca informació per passar de una a l'altra [11]. Reformulem doncs el problema en termes probabilístics: tenim $\mathcal{X} = (X_1, X_2, \dots, X_{n_X})$ i $\mathcal{Y} = (Y_1, Y_2, \dots, Y_{n_Y})$ dues particions dels vèrtexs d'un graf, amb n_X i n_Y clústers cada una, i sigui n el nombre de vèrtexs del graf. Considerem les variables aleatòries X i Y, que, per a cada vèrtex del graf, prenen com a valor l'índex i del clúster al que pertany, en la partició $\mathcal{X}$ i $\mathcal{Y}$, respectivament. Així doncs, els rangs de X i Y són $R_X = \{1, 2, \dots, n_X\}$ i $R_Y = \{1, 2, \dots, n_Y\}$.

Aleshores, tenim que $P(x) = P(X = x) = n_x^X/n$ i $P(y) = P(Y = y) = n_y^Y/n$, amb n_x^X i n_y^Y el nombre de vèrtexs als clústers X_x i Y_y de les particions $\mathcal{X}$ i $\mathcal{Y}$, respectivament, i, si n_{xy} és la quantitat de nodes compartits per les comunitats X_x i Y_y, tenim a més que la distribució conjunta de X i Y és $P(x, y) = P(X = x, Y = y) = n_{xy}/n$. Amb aquests preliminars podem definir la informació mútua de les variables aleatòries X i Y, que mesura quan podem saber sobre Y si coneixem a X, i viceversa:

$$I(X, Y) = \sum_{x=1}^{n_X} \sum_{y=1}^{n_Y} P(x, y) \log \frac{P(x, y)}{P(x)P(y)}$$

Aquesta mesura aplicada al nostre cas té però errors fonamentals: totes les particions formades a partir d'una partició donada $\mathcal{X}$ subdividint algunes de les seves comunitats en comunitats addicionals tindrien la mateixa informació mútua. Per això es fa servir la informació mútua normalitzada de dues particions $\mathcal{X}$ i $\mathcal{Y}$, formulada així:

$$I_{norm}(\mathcal{X}, \mathcal{Y}) = \frac{2I(X, Y)}{H(X) + H(Y)},$$

on $H(X) = -\sum_x P(x) \log P(x)$ és la denominada entropia de Shannon de X. Aquesta mesura pren valors entre 0 i 1, on el 0 representa que les dues estructures de comunitats són totalment diferents, com quan una de les dues consta d'una sola comunitat formada per tots els nodes i l'altra per varies, i l'1 s'assoleix quan són exactament iguals.

A nivell pràctic, per a la implementació en Python d'aquesta mesura s'ha fet servir la proposta de Danon, Díaz-Guilera, Duch, i Arenas [8]. Es defineix una matriu de confusió **N**, on les files corresponen a les comunitats de la partició $\mathcal{X}$, que serà la definida a priori en la creació dels diferents grafs aleatoris, i les columnes a les comunitats de la partició $\mathcal{Y}$, que serà la trobada per l'algoritme corresponent. Els elements N_{ij} de **N** són la quantitat de nodes de la comunitat X_i que també es troben a la comunitat Y_j. Aleshores:

$$I_{norm}(\mathcal{X},\mathcal{Y}) = \frac{\sum_{i=1}^{n_X}\sum_{j=1}^{n_Y} N_{ij}\log\frac{N_{ij}n}{N_iN_j}}{\sum_{i=1}^{n_X} N_i\log\frac{N_i}{n} + \sum_{j=1}^{n_Y} N_j\log\frac{N_j}{n}},$$

, on N_i i N_j són la suma dels valors de la fila i, i de la columna j, respectivament, de **N**, és a dir, $N_i = n_i^X$, i $N_j = n_j^Y$. Com es pot veure, ambdues formulacions de I_{norm} són equivalents.

Una vegada implementat el càlcul de la informació mútua normalitzada, s'ha aplicat cadascun dels algoritmes als 25 grafs aleatoris generats de cada tipus, calculant la mitjana de la informació mútua normalitzada de la partició trobada respecte la predefinida en la construcció del graf, per als 25 grafs i per a cada algoritme. A l'apèndix s'inclouen els programes en Python utilitzats.

2.2 Algoritmes de detecció de comunitats

Algoritme de Girvan-Newman

Aquest algoritme, introduït per Girvan i Newman, es basa en la centralitat d'intermediació per arestes. Estrada [9] explica com aquest concepte és una extensió de la centralitat d'intermediació per nodes ja definida, a les arestes d'un graf, i es defineix com la suma de tots els camins més curts entre dos nodes qualsevol del graf que passen per una aresta determinada, dividida entre la quantitat total de camins més curts. La idea és doncs que les arestes connectant comunitats diferents seran les que tindran un valor més elevat per a aquesta centralitat.

El primer pas d'aquest algoritme és doncs calcular la centralitat d'intermediació per a totes les arestes del graf. A continuació, es treu la que pren un valor més alt, o qualsevol d'elles si diverses assoleixen el màxim. A partir d'aquí es repeteix aquest procés, re-calculant la centralitat d'intermediació per a totes les arestes a cada iteració, fins que no queden connexions al graf. A través d'aquestes iteracions es construeix un dendograma, és a dir, un arbre on cada nivell representa una partició en comunitats diferent del graf, de la següent manera: la línia més inferior consisteix en n punts representant cadascun dels n nodes del graf. Aleshores, a la línia de sobre es representa la unió entre els dos nodes connectats per l'aresta que s'ha tret a la última iteració, simbolitzant $n-1$ comunitats, i així successivament s'afegeix a cada nivell l'aresta corresponent, fins a la primera línia, que representa una sola comunitat amb tots els nodes.

D'aquesta manera, aquest algoritme presenta un rang de possibles comunitats, més que una estructura de comunitats concreta, i cal doncs un criteri per escollir quina volem. Existeixen diverses possibilitats, però en el cas d'aquest treball s'ha fet servir simplement la modularitat. Així doncs, es selecciona la partició en comunitats que té la modularitat màxima com la òptima.

L'algoritme de Girvan-Newman és ja un clàssic en el camp de la detecció de comunitats, i és fa servir habitualment com a referència. Tot i això, és força lent ($\mathcal{O}(nm^2)$ [20], on n és la quantitat de nodes i m d'arestes), degut al càlcul de la centralitat d'intermediació per arestes a cada iteració.

Algoritme de Louvain

Com molts altres, l'algoritme de Louvain, desenvolupat per Blondel, Guillaume, Lambiotte, i Lefebvre [4], està dirigit a trobar la partició en comunitats del graf que maximitzi la modularitat. Com que maximitzar la modularitat és computacionalment difícil, el seu objectiu és més aviat aproximar-se al màxim, mantenint un cost computacional baix (És $\mathcal{O}(m)$ [20], on m és la quantitat d'arestes del graf). De tots els algoritmes basats en maximitzar aquest valor, sembla ser un dels més eficients a l'hora de fer-ho. A més, té en compte els pesos de les arestes, si estan definits, i proporciona diferents estructures de comunitats ordenades jeràrquicament, per la qual cosa permet detectar també sub-comunitats. A continuació se'n detallen els passos.

L'algoritme es divideix en dues fases, que es repeteixen iterativament. En la primera fase, es considera que cada node del graf forma una comunitat diferent. A partir d'aquí, per a cada node es calcula el guany en modularitat que suposaria extreure'l de la seva comunitat i afegir-lo a la comunitat de cadascun dels seus veïns. Si algun d'aquests guanys és positiu, es selecciona el màxim i s'afegeix el node a la comunitat corresponent, sinó, se'l manté a la seva comunitat original. Es repeteix aquest procés fins que cap millora en la modularitat és possible, i aleshores acaba la primera fase. Part de l'eficiència de l'algoritme es basa en les formules usades per computar els guanys de modularitat al moure un node aïllat a una comunitat, o un node formant part d'una comunitat en una altra.

A partir d'aquí, la segona fase consisteix en la creació d'un graf, on cada node representa cadascuna de les comunitats trobades a la fase 1. Les arestes entre nodes són doncs arestes amb pesos, corresponents a la suma de tots els pesos de les arestes entre cadascun dels nodes d'una comunitat i els de l'altra, que són 1 si el graf no té pesos a les arestes. La suma dels pesos de les arestes intra-comunitat s'afegeix com a per a una auto-aresta en aquest comunitat. Amb aquest graf construït es torna a la fase 1, i així successivament, fins que a la fase 2 ja no es produeix cap canvi, la qual cosa indica que s'ha arribat al màxim de modularitat desitjat. S'ha elaborat doncs una estructura jeràrquica de comunitats, amb la construcció de comunitats de comunitats, i amb, al primer nivell, les comunitats desitjades.

Algoritme MiraCom

Aquest algoritme, introduït per Comellas i Miralles [7], fa servir una mesura alternativa a la modularitat per trobar les comunitats del graf, i el seu cost és $\mathcal{O}(mn)$, on m és la quantitat d'arestes i n de nodes del graf. Es tracta del grau mig normalitzat d'un conjunt de nodes: Si V és el conjunt format per tots els nodes del graf, i $C \subset V$ n'és un subconjunt,

$$n\_avgdeg(C) = \frac{\sum_{v_i \in C} \frac{\delta_C^{in}(v_i)}{\delta(v_i)}}{|C|}.$$

Aquesta mesura pren doncs valors entre 0 i 1, ja que és la mitjana de la proporció d'arestes dins del conjunt C, respecte el grau total, per a tots els nodes de C. Així doncs, ens ofereix una mesura del grau en que C forma una comunitat compacta o no.

L'objectiu de l'algoritme MiraCom és doncs trobar el conjunt de comunitats del graf que maximitzi el seu grau mig normalitzat, i ho realitza de forma determinista, constructiva, i fent servir informació local de la xarxa. Esquemàticament, el procediment es basa en formar pre-clústers a partir d'un node i els seus veïns, seleccionant-los de forma a que compleixin unes determinades propietats, i ajuntar aquests pre-clústers sota certes condicions. Es comença creant una llista L amb tots els vèrtexs del graf en ordre decreixent en funció del seu grau. A partir d'aquí, iterativament, es pren el primer node v d'aquesta llista que tingui com a

mínim la meitat de les seves adjacències a L, i es seleccionen, d'entre els seus veïns, els que tenen també com a mínim la meitat de les seves adjacències a L, i, d'entre aquestes connexions, com a mínim la meitat amb veïns de v o amb el mateix v. Amb aquest vèrtexs, que s'extreuen de L, es forma el pre-clúster C_{pre}. A continuació es comprova si es pot afegir C_{pre} a algun clúster C_i ja existent: el grau mig normalitzat de la unió dels dos clústers ha de ser més gran o igual que el grau mig normalitzat de cadascun dels clústers per separat, i la quantitat d'adjacències entre ells ha de ser més gran que $|C_{pre}| \cdot |C_i| \cdot avgdeg(G)/(|V|-1))$, és a dir, que la quantitat esperada de connexions entre els dos clústers en funció del grau mig del graf. Si no es pot, i C_{pre} té més de dos nodes, es crea un nou clúster amb ell. Es repeteix aquest procés fins que no quedin nodes a la llista L, o fins que el C_{pre} obtingut sigui buit. En aquest segon cas, s'afegeix cadascun dels nodes restants a L al clúster amb el que tinguin més connexions.

Algoritme Infomap

En aquest cas, es tracta d'un algoritme força diferent als anteriors, basat en teoria de la informació, i desenvolupat per Rosvall i Bergstrom [25]. El seu cost és $\mathcal{O}(m)$ [20], on m és la quantitat d'arestes del graf. Funciona per grafs dirigits i amb pesos, i, resumidament, busca l'estructura de comunitats que permeti codificar el més eficientment possible un passeig aleatori per el graf. La idea es assignar codis formats per uns i zeros a cada comunitat, i, dins de cadascuna, a cadascun dels seus nodes, podent-se repetir aquests entre nodes de diferents comunitats. Així, enlloc d'assignar un codi diferent a cada node, s'aprofita l'estructura dels clústers per a aconseguir, en mitjana, codis més curts per a un passeig aleatori qualsevol, ja que és estadísticament més probable que el camí es quedi més temps dins de les comunitats que entre elles. Es necessita doncs, a més d'un codi diferent per a cada node de la comunitat, un codi que indiqui l'entrada i la sortida de la comunitat, diferents per a cadascuna.

A partir d'aquí, per trobar la partició del graf que minimitza la llargada mitjana del codi necessari per a descriure un passeig aleatori qualsevol, l'algoritme realitza una cerca computacional, ja que per xarxes que no siguin molt petites és molt lent comparar totes les particions possibles. Computa doncs la fracció de temps que passa el passeig aleatori en cada node amb el mètode de les potències, i amb aquests valors realitza una cerca determinista de tipus greedy, que es refina més endavant. A més, s'introdueix una "probabilitat de tele-transportació", és a dir, la probabilitat de que el procés salti, de cop, a un altre node aleatori del graf, que es fixa a 0.85, la qual cosa resta una mica de fiabilitat a l'algoritme però n'augmenta la rapidesa.

Com es pot veure, l'aproximació al problema de la detecció de comunitats que hi ha darrere l'algoritme Infomap és conceptualment diferent a la dels algoritmes descrits anteriorment, sobretot el Louvain i el Miracom. L'Infomap es basa, en certa manera, en els fluxos d'informació que descriu la xarxa, i en canvi fent ús de la modularitat, per exemple, s'atén a l'estructura topològica mostrada per les relacions en aquesta. Aquesta diferència no és només conceptual, ja que es poden trobar exemples de grafs on l'estructura de comunitats trobada per l'Infomap proporciona una modularitat baixa, molt més que la que trobaria un algoritme que busqui la maximització d'aquesta com el Louvain, i en aquest segon cas però la llargada mitjana per pas, en bits, d'un passeig aleatori, seria molt més llarga. Així doncs, pot ser important tenir en compte què modelitza la xarxa en qüestió, alhora d'escollir el tipus d'algoritme a usar per trobar-ne les comunitats: si aquesta vol representar fluxos entre els diferents actors, té més sentit fer servir l'Infomap, en canvi, si els enllaços representen relacions bidireccionals i no es té tant en compte el moviment d'una determinada quantitat entre els nodes, seria més adequat fer servir el Louvain, per exemple.

2.3 Resultats

Es presenta a continuació dues taules amb les mitjanes de la informació mútua normalitzada per a cada tipus de graf i cada algoritme. Per als grafs amb comunitats estàndard les etiquetes 'Iguals' i 'Diferents' fan referència a si les comunitats tenen totes el mateix nombre de nodes o no. La proporció que es mostra a l'apartat dels grafs regulars amb comunitats correspon al valor mig de δ^{out}/δ^{in} per a cada tipus de graf.

Grafs amb comunitats estàndard						
	(0.25,0.08)		(0.6,0.08)		(0.25,0.01)	
	Iguals	Diferents	Iguals	Diferents	Iguals	Diferents
Louvain	0.46	0.35	1.00	0.95	0.99	0.87
Miracom	0.04	0.15	1.00	0.90	0.89	0.87
Girvan-Newman	0.43	0.57	1.00	0.79	0.99	0.94
Infomap	0.00	0.00	1.00	1.00	0.99	0.97

	Grafs LFR		Grafs regulars amb comunitats	
	$\mu = 0.2$	$\mu = 0.5$	$1/14$	$7/8$
Louvain	1.00	0.89	1.00	0.61
Miracom	1.00	0.65	1.00	0.02
Girvan-Newman	1.00	0.73	1.00	0.50
Infomap	1.00	0.08	1.00	0.00

Discussió

Per començar, destaca la varietat en els resultats obtinguts per a cada algoritme, en funció del tipus de graf aleatori utilitzat per realitzar els tests. Si ens fixem només en els graf amb comunitats estàndard, podem observar com l'algoritme Infomap és el que funciona millor per els grafs amb comunitats ben definides, és a dir amb $(p_{in}, p_{out}) = (0.6, 0.08)$ o (0.25,0.01), però en canvi quan aquestes no ho estan tant, vorejant el límit en que es possible la detecció, sembla ser que considera tot el graf com una sola comunitat. Analitzant com es comporta aquest algoritme a l'estudi de Lancichinetti i Fortunato [20], la conclusió és que amb una variació molt petita de la proporció d'arestes internes i externes a les comunitats, corresponent a un interval per al paràmetre μ d'entre $\mu = 0.4$ i $\mu = 0.45$, passa de detectar-les a la perfecció a, bruscament, fer-ho molt malament, per a valors de μ amb els que altres algoritmes tenen encara resultats acceptables. Això si, la seva precisió no es veu afectada per les diferències establertes en la mida de les comunitats, ni per la major o menor densitat d'arestes del graf, com si passa amb els altres algoritmes. Passant ara als grafs LFR observem un efecte similar: l'Infomap passa de detectar les comunitats a la perfecció a fer-ho pèssimament, quan encara estan prou definides perque altres algoritmes les detectin força bé. A més, en aquest cas resulta especialment interessant la comparació amb l'article de Lancichinetti et al. [20], ja que en aquest estudi, amb grafs aleatoris LFR de 1000 vèrtexs l'Infomap detecta les comunitats perfectament per a, com a mínim, $\mu = 0.6$, i amb 5000 vèrtexs o fa per a $\mu = 0.7$, mentre que en el nostre cas amb $\mu = 0.5$ ja no les troba gens bé. Finalment, amb els grafs regulars amb comunitats els resultats obtinguts són equivalents als obtinguts amb els altres grafs aleatoris, així que no ofereixen més profunditat analítica. Es pot concloure doncs que és un algoritme que funciona millor com més grans siguin les xarxes, i ho fa, podríem dir, de forma molt binària: o troba les comunitats existents a la perfecció o les confon totes. Aquest segon fet segurament és conseqüència de que considerar tots els nodes com una sola comunitat

suposa una codificació dels camins aleatoris més curta que la corresponent a la majoria d'estructures de comunitats possibles, menys la correcta i alguna més. És doncs un algoritme molt útil per a xarxes grans, donat el seu baix cost, però si les xarxes són més petites i l'estructura de comunitat que extreu està formada per aproximadament una sola comunitat, resulta convenient fer servir un altre algoritme per a contrastar.

L'algoritme Louvain, per una altra banda, és el que, en mitjana, i considerant els grafs aleatoris utilitzats, funciona millor. Veiem com detecta igual de bé o similarment les comunitats en els casos en els que l'Infomap ho feia molt bé, però a més per els casos en que aquest altre algoritme fallava es manté en valors força acceptables. Això si, per els grafs amb comunitats estàndard sembla com l'estructura de comunitats (16,16,32,64) disminueix sensiblement la seva precisió respecte la (32,32,32,32), però tot i així la diferència no és molt gran. Per als grafs LFR amb $\mu = 0.5$ la informació mútua normalitzada mitjana que obté és molt bona, més que per als grafs amb comunitats estàndard amb $(p_{in}, p_{out}) = (0.25, 0.01)$ i comunitats diferents, la qual cosa resulta curiosa perquè en aquest segon cas el paràmetre μ corresponent és de 0,10. Així doncs, quan $(p_{in}, p_{out}) = (0.25, 0.08)$, corresponent ara si a $\mu \cong 0.5$, detecta molt pitjor les comunitats que pels grafs LFR. Veiem com aquesta major precisió per als grafs LFR respecte els grafs amb comunitats estàndard es repeteix per a tots els algoritmes, i segurament és deguda a la major grandària de les comunitats en els segons. Comparant de nou amb l'anàlisi de Lancichinetti i Fortunato [20], veiem com els resultats hi coincideixen força, tot i que en el seu cas els grafs amb comunitats estàndard que fan servir tenen un grau mig per node més alt, i els LFR molt més nodes. Resulta interessant veure, a més, com els resultats que ha obtingut el Louvain són més bons per als grafs regulars amb comunitats que per els estàndard. Com a conclusió global respecte aquest algoritme podem dir que és molt robust, en el sentit de que funciona força bé independentment de la mida del graf, de com siguin les seves comunitats, o d'altres característiques d'aquest. Aquest fet és força lògic si tenim en compte que es basa en la maximització de la modularitat: en certa manera, hi ha generalment moltes estructures de comunitats possibles per a les quals aquesta quantitat serà semblant, i força més alta que la corresponent a una sola comunitat formada per tots els nodes, mentre aquestes estiguin mínimament ben definides, per la qual cosa es veu la diferència amb l'Infomap.

Passant ara a l'algoritme Girvan-Newman, veiem com els resultats que ha obtingut segueixen uns patrons clarament diferents als dos algoritmes que ja s'han analitzat, i que a més són força irregulars. Així, fixant-nos per començar en els grafs amb comunitats estàndard veiem com quan $(p_{in}, p_{out}) = (0.25, 0.08)$, funciona millor quan l'estructura de comunitats és (16,16,32,64) que quan és (32,32,32,32), però quan és (0.6,0.08) funciona clarament millor en el segon cas. Podríem dir que quan les comunitats estan poc definides és més precís amb la primera estructura de comunitats, però quan ho estan força és més precís amb tots els clústers iguals. Sembla a més que sigui més adequat com menys connexions tingui el graf, destacant la baixa precisió quan $(p_{in}, p_{out}) = (0.6, 0.08)$, en comparació amb els altres algoritmes. Aquest fet es deu probablement al propi funcionament de l'algoritme: com més densa en arestes és una xarxa, menys rellevància hi tenen els camins més curts entre nodes, ja que n'hi hauran segurament diversos. A més, hi hauran molts camins amb quasi la mateixa llargada que els més curts. Si, a més, la meitat del graf el conforma una sola comunitat, com és el cas quan les comunitats tenen mides diferents, molts dels camins més curts estaran a dins d'aquesta mateixa comunitat i per tant no permetran distingir entre diferents clústers. Així doncs, amb xarxes poc denses ($(p_{in}, p_{out}) = (0.25, 0.08)$ o (0.25,0.01)) resulta ser un bon algoritme, en mitjana millor que el Louvain i l'Infomap. Tot i això, amb els graf LFR és substancialment pitjor que el Louvain, degut segurament al major grau mig en aquests grafs, igual que amb els grafs regulars. Si comparem amb l'estudi de Lancichinetti i Fortunato [20] sembla reforçar-se l'explicació que s'ha donat per al millor o pitjor funcionament de l'algoritme de Girvan-Newman, ja que en aquest article, amb grafs similars als considerats en aquest treball, però grau mig per node força més alt, els resultats són pitjors. La conclusió general per a aquest algoritme és doncs que pot ser útil i oferir molt bons resultats en certs

casos, com en xarxes poc denses, però en general és poc robust, i el seu alt cost en dificulta l'aplicació quan el graf a tractar arriba a una mida mitjana.

Finalment, el Miracom és l'algoritme que ha donat pitjors resultats, ja que per als tipus de grafs amb els que l'Infomap ha obtingut un 0 de mitja per a la informació mútua normalitzada obté valors una mica més alts, però sempre per sota dels altres dos algoritmes, i quan les comunitats estan més clarament definides tampoc funciona especialment bé. Així, l'Infomap i el Girvan-Newman, tot i ser poc precisos per a certs tipus de grafs, ho són força per d'altres, per a la qual cosa resulten útils, mentre que el Miracom no aporta valor afegit en cap de les tipologies de graf aleatori considerada.

3. Centralitat i situació a la comunitat

3.1 Procediment

L'objectiu d'aquesta part del treball és comprovar, en una xarxa amb estructura de comunitats, quina és la correlació entre la posició d'un node a la seva comunitat, i la seva centralitat respecte el total del graf. A l'hora d'operativitzar el confós concepte de la "posició"d'un node a la seva comunitat, s'ha escollit simplement la proporció $r = \delta^{out}/\delta^{in}$ del node. És a dir, que es vol veure si els nodes més centrals a nivell de tot el graf són els que tenen més connexions cap a l'exterior de la seva comunitat respecte les que tenen amb altres nodes de la seva comunitat, el que es tradueix en un valor de r més petit. Serien doncs els nodes que exerceixen de pont entre una comunitat i una altra, i que per tant controlen els fluxos entre aquestes. Una altra opció hagués sigut fer servir la centralitat dels vèrtexs respecte el subgraf format per la seva comunitat, però l'avantatge del ràtio r és que no té en compte el nombre d'arestes del node, i per tant inclou la possibilitat de que vèrtexs amb poques connexions i per tant poca centralitat al subgraf, si siguin centrals a nivell de tot el graf.

S'han fet servir diferents mesures de centralitat, per a veure les diferències proporcionades per cada una: la centralitat d'intermediació, la de proximitat, la de 'comunicabilitat' o subgraf, la de vector propi ("eigenvector centrality"), i les versions en flux de la centralitat d'intermediació i de proximitat, definides totes a la introducció d'aquest treball. S'ha decidit no utilitzar la popular mesura PageRank, que és la que utilitza l'algoritme del buscador Google per determinar la importància d'una pàgina web, tot i haver-la incorporat inicialment, degut a que té més sentit per a grafs dirigits, i amb grafs no dirigits dona resultats similars a la centralitat de vector propi.

Per una altra banda, s'han utilitzat els tres tipus de grafs aleatoris definits, per a poder calcular mitjanes que aportin robustesa als resultats. En el cas dels grafs amb comunitats estàndard i dels LFR l'objectiu és simular grafs reals. En canvi els grafs regulars amb comunitats no representen xarxes extretes de dades reals més que en casos molt particulars, però ens permeten veure quina seria la correlació si no tenim en compte l'efecte de la distribució de graus d'un graf. Es pot comprovar doncs la importància de que un node exerceixi o no de pont entre comunitats respecte la seva centralitat, en condicions iguals en termes del nombre de connexions respecte la resta de nodes. Finalment, també s'ha aplicat l'anàlisi a una serie de xarxes extretes de dades reals: s'han fet servir tres xarxes d'aeroports, d'Estats Units, Àsia, i Nord-Amèrica [18], una xarxa de emails enviats entre professors de la Universitat Rovira i Virgili [17], una formada per músics de jazz on les arestes es formen entre músics que han tocat alguna vegada junts [14], una xarxa de dofins creada a partir del seu contacte freqüent [1], una entre practicants de *windsurf* [3], i finalment la xarxa de terroristes de l'11-S [2]. En tots els casos es tracta de grafs relativament petits, d'entre 43 i 1133 nodes, no dirigits, i si tenien pesos s'han omès. A més, tots són connexos.

El procediment és el següent: es calculen les diferents mesures de centralitat per a cadascun dels nodes de la xarxa, per una banda, i la proporció r per a cada node, per l'altra, fent servir les comunitats predefinides en el cas de grafs aleatoris, o les comunitats trobades amb l'algoritme Louvain en el cas de les xarxes extretes de dades reals. A continuació es calcula la correlació per a cada parell de llistes, formades una per una de les mesures de centralitat de cada node, i l'altra per el valor r de cada node, amb cada parell corresponent a una centralitat diferent. Si es tracta de grafs aleatoris, es calcula la mitjana de cada correlació per als 25 grafs de cada tipus que s'han generat.

3.2 Resultats

S'inclouen a continuació les taules amb les correlacions resultants del procediment explicat. Les etiquetes 'I' i 'D' a la taula dels grafs amb comunitats estàndard fan referència a si les comunitats són iguals en els grafs tractats (estructura (32,32,32,32)), o si són diferents (estructura (16,16,32,64)), i els noms de les columnes fan referència al tipus de centralitat considerat. Per una altra banda, s'ha fet servir la mateixa notació que a la presentació dels resultats de la secció 2 per a caracteritzar el tipus de graf amb comunitats regulars.

Grafs amb comunitats estàndard							
		Flux de proximitat	Flux d'inter-mediació	Vector propi	Proximitat	Intermediació	Subgraf
(0.25,0.08)	I	-0.00	0.02	-0.01	0.08	0.05	-0.01
	D	-0.37	-0.28	-0.38	-0.31	-0.23	-0.37
(0.6,0.08)	I	0.36	0.58	0.31	0.46	0.62	0.31
	D	-0.72	-0.23	-0.68	-0.66	-0.25	-0.67
(0.25,0.01)	I	0.07	0.46	-0.00	0.45	0.48	-0.12
	D	-0.34	0.13	-0.29	-0.17	0.11	-0.29

Grafs LFR						
	Flux de proximitat	Flux d'inter-mediació	Vector propi	Proximitat	Intermediació	Subgraf
$\mu = 0.2$	-0.05	0.06	-0.05	0.05	0.06	-0.05
$\mu = 0.5$	-0.05	-0.04	-0.04	-0.02	-0.03	-0.04

Grafs regulars amb comunitats						
	Flux de proximitat	Flux d'inter-mediació	Vector propi	Proximitat	Intermediació	Subgraf
1/14	0.95	0.98	0.00	0.87	0.94	-0.96
7/8	0.46	0.46	0.01	0.41	0.47	-0.52

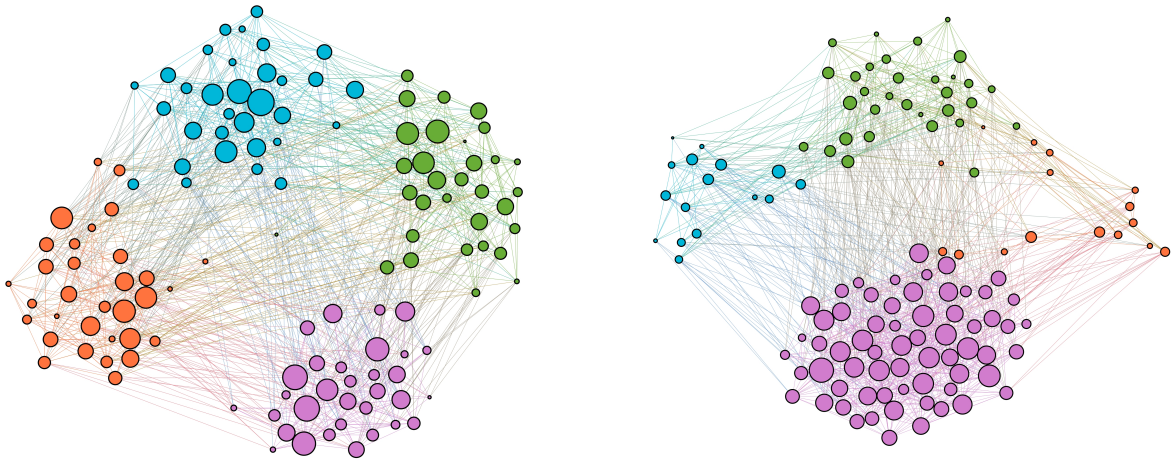
Xarxes reals						
	Flux de proximitat	Flux d'inter-mediació	Vector propi	Proximitat	Intermediació	Subgraf
Aeroports USA	0.52	0.47	0.53	0.59	0.40	0.48
Aeroports Asia	0.46	0.46	0.01	0.41	0.47	-0.52
Aeroports N-A	0.41	0.61	0.37	0.53	0.56	0.29
Email	0.29	0.35	0.18	0.40	0.33	0.08
Jazz	0.23	0.39	0.34	0.42	0.46	0.33
Dofins	0.47	0.51	0.26	0.66	0.55	0.08
Windsurf	0.47	0.40	0.54	0.51	0.32	0.53
Terroristes	0.19	0.27	0.14	0.28	0.21	0.16

La primera impressió que es desprèn de les taules és que, per a la majoria de grafs considerats, les

correlacions amb les mesures de centralitat són força altes. Destaquen els grafs LFR, i els grafs amb comunitats estàndard on aquestes són iguals i poc definides, per als quals la correlació amb totes les mesures de centralitat és propera al zero, i així doncs significativa. Un altre patró interessant que es detecta fàcilment en els grafs amb comunitats estàndard és el signe positiu de la correlació quan les comunitats són iguals, i negatiu quan són diferents. En canvi, per a les xarxes reals les correlacions són sempre positives, menys en algun cas molt concret. També destaquen les altíssimes correlacions per als grafs regulars amb comunitats ben definides, corresponent així amb el que s'havia previst.

La raó per la qual la proporció r dels nodes dels grafs LFR no es correlaciona amb les diferents mesures de centralitat és ben senzilla: per construcció, aquests grafs tenen un valor per al coeficient μ molt similar entre els nodes, i com que $\mu = \delta^{out}/(\delta^{in} + \delta^{out})$ i $r = \delta^{out}/\delta^{in}$, aquest segon nombre no varia prou com per correlacionar-se amb la centralitat. Així doncs, es pot dir que forçar els nodes d'aquest tipus de graf aleatori a tenir una proporció quasi fixa de connexions internes i externes a la seva comunitat, permet, per una banda, assegurar que la comunitat que se'ls hi assigna sigui realment la comunitat a la que pertanyen, però per l'altra provoca que aquests grafs no comparteixin una característica, que, pel que es pot observar als resultats presentats, probablement sigui present a les xarxes complexes en general. Es pot concloure que el propi caràcter de l'estructura de comunitats en una d'aquestes xarxes, com a entitats sense una definició que en permeti realment la unicitat, és doncs una característica intrínseca d'aquestes, ja que, quan s'intenta sortejar aquest obstacle amb grafs aleatoris com els LFR sorgeixen altres incorreccions respecte la realitat.

Per una altra banda, en el cas dels grafs amb comunitats estàndard amb comunitats iguals i $(p_{in}, p_{out}) = (0.25, 0.08)$, el fet de que no hi hagi correlació segurament és degut a que no es distingeixen prou. Resulta interessant doncs fixar-se en com apareixen correlacions considerables, i negatives, per a aquests mateixos graf amb comunitats diferents. Sembla ser que, si les comunitats estan poc definides, i són iguals, la situació d'un vèrtex en el graf no tingui especial rellevància a l'hora de determinar la seva centralitat, però quan existeix una comunitat gran, acaparant la meitat de la xarxa, amb altres tres comunitats secundàries, si que és important tenir més connexions amb la teva comunitat que amb les altres. Amb els gràfics següents s'il·lustren aquestes observacions.

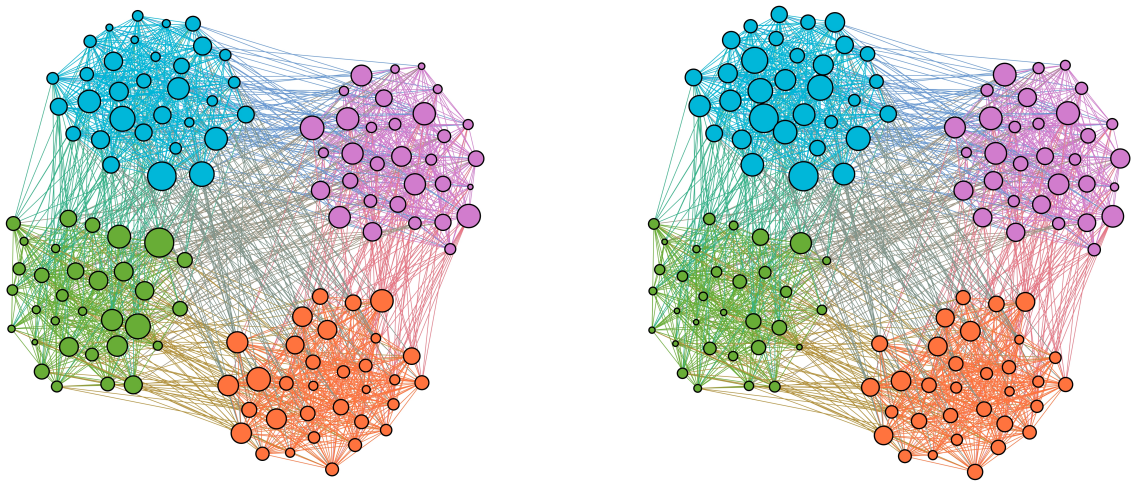


(a) (b)

Figura 1: Graf amb comunitats estàndard amb $(p_{in}, p_{out}) = (0.25, 0.08)$. El color dels nodes representa la comunitat i la mida la seva centralitat de vector propi. A l'esquerra el graf té quatre comunitats de 32 nodes, i a l'esquerra dues comunitats de 16 nodes, una de 32 i una de 64.

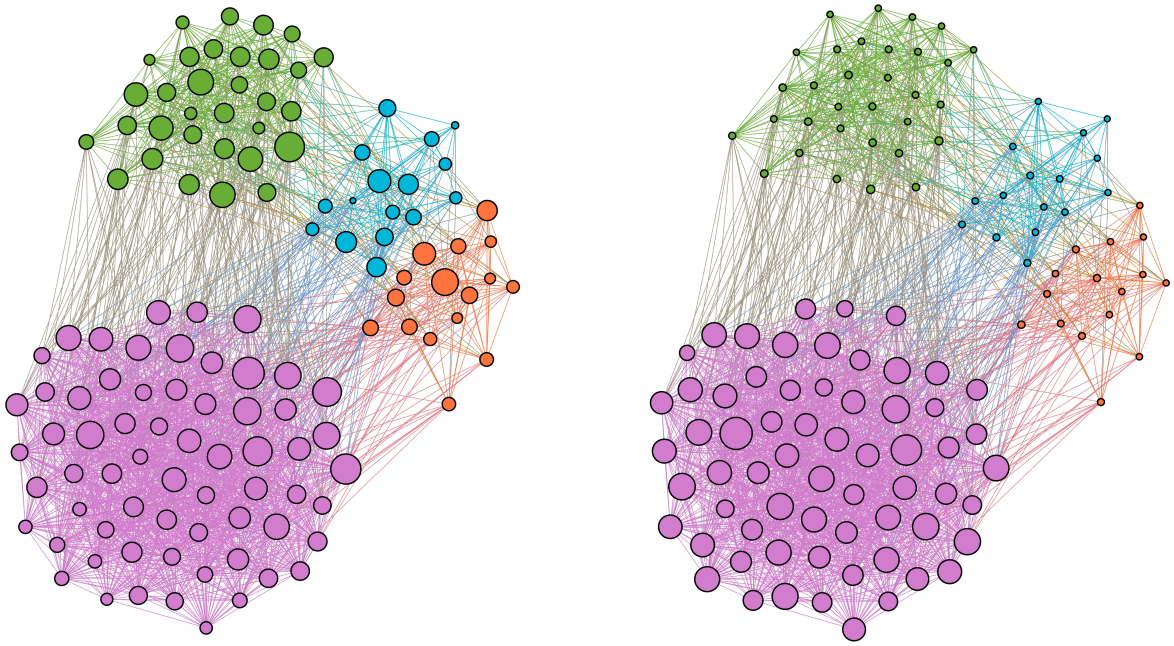
Veiem com a la Figura 1.a) els nodes més grans es troben dispersos de forma força uniforme per la seva comunitat, i no es distingeix cap patró concret en la seva distribució, mentre que a la Figura 1.b), pel que fa a la comunitat més gran, els nodes més petits són precisament els que hi estan situats al seu exterior, i els més grans al seu interior. Les dues comunitats de 16 nodes estan poc definides i costa determinar un patró a partir de com s'ha representat la xarxa, però a la comunitat de 32 nodes sembla més aviat que els nodes externs són els més centrals. Es conclou que la correlació negativa entre r i la centralitat de vector propi és deguda a la presència d'una comunitat que agrupa la meitat de la xarxa, i per tant amb vèrtexs molt més centrals que a la resta del graf. Cal tenir en compte però una consideració: els grafs amb comunitats estàndard amb comunitats diferents, tenen, per construcció, coeficients μ mitjans per als nodes de cada comunitat força variats. Per exemple, quan $(p_{in}, p_{out}) = (0.6, 0.08)$, a la comunitat de 64 nodes μ val en mitjana 0.12, mentre que a la de 32 val 0.29 i a les de 16 val 0.5. Amb aquestes dades s'expliquen les altes correlacions negatives que apareixen a la taula per a aquests grafs: els nodes més centrals són els de la comunitat més gran, i per la forma en que estan construïts els grafs, aquests tenen moltes més connexions internes, respecte les externes, que la resta de nodes. Aquest fet explica doncs, com a mínim en part, perquè a les xarxes reals no apareixen quasi correlacions negatives, i sembla que desvela un error, pel que fa al trasllat a la realitat, dels grafs amb comunitats estàndard: les comunitats més grans d'aquests grafs estaran sempre més densament connectades que les altres.

Si s'analitzen, a més, les diferències entre grafs amb comunitats estàndard amb comunitats diferents i amb comunitats iguals, pel que fa a les diferents mesures de centralitat considerades, veiem com les centralitats de proximitat, de flux de proximitat, de vector propi, i de subgraf, cobren més importància per a comunitats diferents que quan són iguals, i en canvi amb la centralitat de flux d'intermediació i la d'intermediació passa el contrari. Es pot interpretar que, amb l'estructura de comunitats (32,32,32,32), els nodes que fan de frontera entre comunitats, al acaparar els camins passant d'una a l'altra són molt centrals segons aquestes dues últimes mesures, però quan aquesta és (16,16,32,64), no són tampoc els nodes més interns els que es fan més d'intermediari entre camins, ja que una part important dels camins segueixen essent entre comunitats.



(a) (b)

Figura 2: Graf amb comunitats estàndard amb $(p_{in}, p_{out}) = (0.6, 0.08)$ nº13, amb 4 comunitats de 32 nodes cadascuna. El color dels nodes representa la comunitat i la mida la seva centralitat d'intermediació en flux, a l'esquerra, i de subgraf, a la dreta.



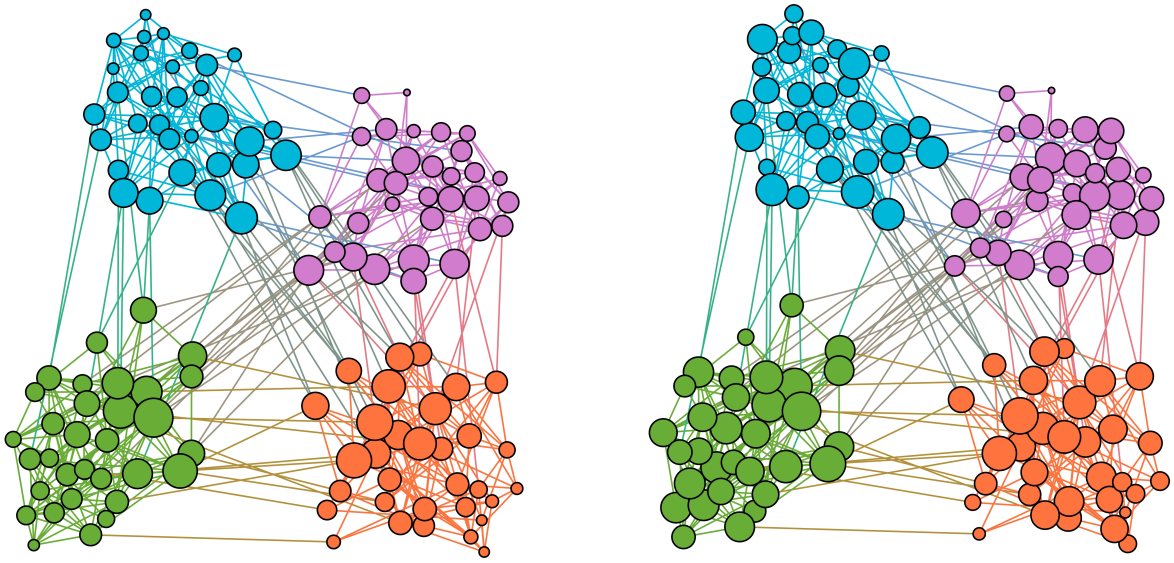
(a) (b)

Figura 3: Graf amb comunitats estàndard amb $(p_{in}, p_{out}) = (0.6, 0.08)$ nº13, i 2 comunitats de 16 nodes, una de 32 i una de 64. El color dels nodes representa la comunitat i la mida la seva centralitat d'intermediació en flux, a l'esquerra, i de subgraf, a la dreta.

Es poden visualitzar a les Figures 2 i 3 les observacions realitzades. Destaquen sobretot les diferències a la Figura 3: a la 3.a), com es compten tots els camins entre parells de nodes en els que es troba el node en qüestió, hi ha nodes a les comunitats més petites tant centrals com a la comunitat més gran. En canvi, en el cas de la centralitat de subgraf a 3.b) la major densitat de connexions a la comunitat de 64 nodes provoca que hi hagin molts més camins, i més curts, que surten d'un node i hi tornen, que a les altres comunitats. També s'observa com els nodes amb un major valor per a aquesta centralitat es troben més aviat a l'interior de la comunitat, i són doncs el que tenen una r més petita, ja que a la centralitat de subgraf les aportacions més grans són realitzades per els cicles més curts, i si es mantenen a la mateixa comunitat, més densa que les altres, ho seran més. En canvi en el cas de la centralitat de flux d'intermediació semblen estar més a l'exterior de la comunitat més gran, per la qual cosa es pot interpretar que la correlació negativa que apareix per als grafs amb $(p_{in}, p_{out}) = (0.6, 0.08)$ i comunitats diferents amb aquesta mesura de centralitat no és deguda a que els nodes més centrals es trobin a l'interior de la comunitat, sinó a que es troben a la comunitat més gran on r és en mitjana més petita. De fet, en aquests mateixos grafs amb $(p_{in}, p_{out}) = (0.25, 0.01)$, la correlació amb la centralitat d'intermediació, i de flux d'intermediació, és positiva.

Veiem com per als grafs amb aquesta última configuració per a (p_{in}, p_{out}), la poca densitat d'arestes entre comunitats provoca diferències interessants en les correlacions obtingudes per a cada mesura de centralitat. Per exemple, amb comunitats iguals la correlació de la centralitat de proximitat en flux és quasi nul·la, però en canvi la de la centralitat de proximitat és força alta. S'interpreta que, si són pocs els nodes amb diverses connexions amb comunitats altres que la seva, aquests es trobaran clarament més a prop de la resta de nodes, tenint en compte només els camins més curts. En canvi, si es tenen en compte tots els

camins, encara que els més curts tinguin més pes, tenir com a veí un node connectat a una altra comunitat ja pot conferir força centralitat. Amb la figura següent es poden visualitzar aquestes observacions.



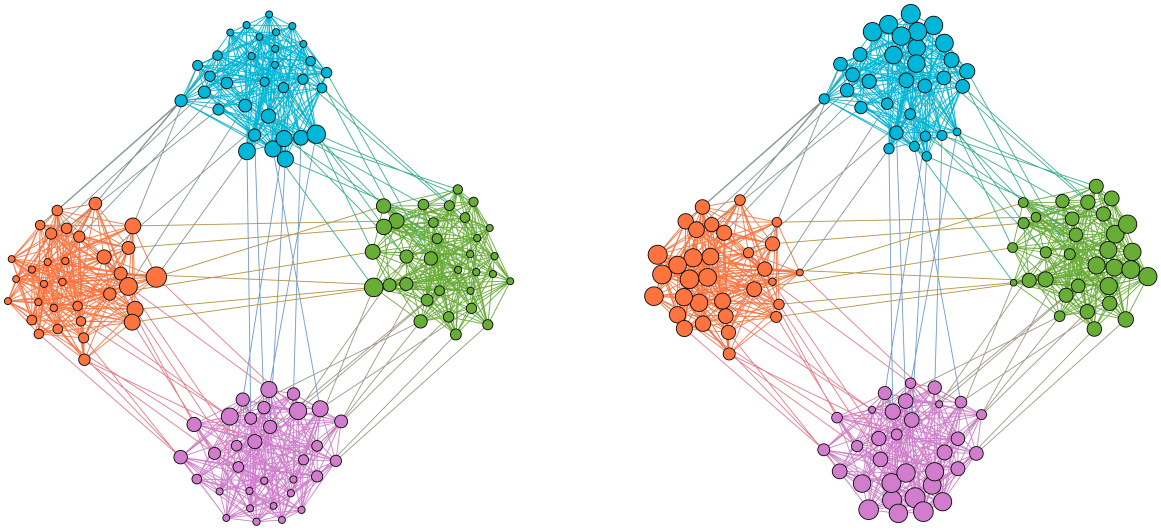
(a) (b)

Figura 4: graf amb comunitats estàndard nº5 amb $(p_{in}, p_{out}) = (0.25, 0.01)$ i 4 comunitats de 32 nodes cadascuna. El color dels nodes representa la comunitat i la mida la seva centralitat de proximitat, a l'esquerra, i la centralitat de proximitat en flux, a la dreta

Pel que fa a la centralitat d'intermediació, i d'intermediació en flux per a aquests mateixos grafs, es pot argumentar perquè les seves correlacions són altes de forma similar. En aquest cas, la versió en flux també presenta correlacions altes, probablement perquè per a que un node exerceixi d'intermediari en un camí, encara que no sigui dels més curts, no té gaire influència ser veí d'un altre node que sigui intermediari, al contrari del que passava per la centralitat de proximitat en flux. D'altra banda, veiem com amb el canvi a comunitats diferents per a aquesta mateixa parella (p_{in}, p_{out}) s'inverteixen les mesures de centralitat que prenen valors més extrems, repetint-se l'esquema anterior segons el qual les correlacions de les centralitats de vector propi i de subgraf s'incrementen força, i les de les dues centralitats d'intermediació disminueixen. La diferència amb el que passava quan $(p_{in}, p_{out}) = (0.6, 0.08)$ resideix doncs en la centralitat de proximitat, que en aquest segon cas tenia un comportament, respecte a la comparació entre comunitats iguals i diferents, similar al de les centralitats de vector propi i de subgraf.

Passant ara als grafs regulars amb comunitats, destaquen clarament les altíssimes correlacions resultants en valor absolut quan les comunitats estan prou ben definides, menys en el cas de la centralitat de vector propi. Veiem com amb aquests grafs la proporció d'arestes internes i externes a la comunitat ens indica amb molta precisió com de central serà el node, essent quasi equivalents les centralitats d'intermediació, proximitat, i les seves versions en flux. En canvi, la centralitat de subgraf es comporta de forma quasi exactament inversa, la qual cosa ens mostra una diferència fonamental entre aquesta mesura de centralitat i les altres quatre. A la figura 5 es poden visualitzar aquestes diferències. En certa manera, es pot interpretar que les quatre primeres centralitats modelitzen fluxos d'informació, o d'una quantitat determinada, a la xarxa, per la qual cosa, si tots els nodes tenen el mateix grau, resulta molt important trobar-se en els punts de comunicació entre comunitats clarament diferenciades per a ser central des d'aquest punt de vista. En canvi, la centralitat de subgraf prioritza que el conjunt de veïns i vèrtexs propers al considerat sigui dens,

de manera a que hi hagin molts camins curts que retornin a aquest vèrtex, i per això són molt més centrals des del seu punt de vista els nodes amb moltes connexions a dins de la comunitat, quan les comunitats estan poc comunicades entre elles. S'entén doncs a la llum d'aquestes observacions encara millor perquè se l'anomena centralitat de subgraf. Destaca a més com la centralitat que presenta amb més freqüència correlacions negatives en el global dels resultats, ja que aquesta només és positiva, per als grafs aleatoris considerats, en el cas dels grafs amb comunitats estàndard quan $(p_{in}, p_{out}) = (0.6, 0.08)$, i amb comunitats iguals. Es pot raonar que en aquest cas és positiva degut a que les connexions amb una comunitat diferent a la del node en qüestió són més freqüents, i aleshores els camins que en surten, passen per una altra comunitat, i hi tornen, són força rellevants en el còmput global.



(a) (b)

Figura 5: graf amb comunitats regular n°0 amb 1 connexió externa a la comunitat en mitjana. El color dels nodes representa la comunitat i la mida la seva centralitat d'intermediació, a l'esquerra, i la de subgraf, a la dreta.

Per una altra banda, la pròpia definició de la centralitat de vector propi causa que, en un graf regular, tots els nodes en tinguin la mateixa, ja que aquesta mesura es basa, al cap i a la fi, en la quantitat de connexions de cada node. Tot i que té també en compte si aquestes connexions són amb nodes també centrals, l'únic criteri que fa servir en última instància és el grau, i es pot interpretar doncs que no és més que una mesura refinada de la més senzilla centralitat basada en el grau. Així doncs, no permet identificar molts dels matisos més enllà d'aquesta quantitat, que si detecten les altres centralitats considerades. Es conclou dels resultats amb grafs regulars amb comunitats, que, en termes de fluxos en una xarxa, una vegada anul·lat l'efecte del grau de cada node, els més centrals seran els que exerceixin de frontera entre comunitats, si aquestes estan prou definides.

Finalment, els resultats per a xarxes reals mostren, en línies generals, correlacions relativament altes, i sempre positives, menys per la xarxa d'aeroports d'Àsia amb la centralitat de subgraf. A continuació es presenta una taula amb, per a cadascuna d'aquestes xarxes, el nombre de comunitats trobades per l'algoritme de Louvain, el percentatge de nodes respecte al total de cada una, per les 10 més grans, i la modularitat de la partició, per a poder així comparar els resultats amb els obtinguts amb els grafs aleatoris.

		Aeroports USA	Aeroports Asia	Aeroports N-A	Email	Jazz	Dofins	Windsurf	Terroristes
# nodes		295	706	940	1133	198	62	43	64
# comunitats		5	12	22	12	4	5	2	5
modularitat		0.30	0.64	0.51	0.54	0.44	0.52	0.25	0.45
% nodes	Comunitat								
	1	37.97	24.79	20.96	21.98	30.81	29.03	53.49	34.38
	2	30.51	16.29	18.19	21.01	29.80	29.03	46.51	31.25
	3	15.93	12.89	12.23	10.86	29.80	19.35		15.63
	4	14.24	11.76	8.94	10.50	9.60	14.52		14.06
	5	1.36	10.20	7.13	8.91		8.06		4.69
	6		9.07	6.49	7.94				
	7		4.82	6.38	6.44				
	8		4.11	5.21	4.85				
	9		2.97	4.15	4.68				
	10		2.12	3.83	2.03				

Per començar, un cop d'ull a la modularitat de la partició en comunitats trobada per a cada xarxa ens indica com les xarxes amb una modularitat més alta no són les que tenen correlacions més altes, i viceversa. Tot i tenint en compte que la modularitat tendeix a augmentar amb la mida d'una xarxa, en aquest cas, degut a que la disparitat en la quantitat de nodes de les xarxes considerades no es excessiva, es pot fer servir com a mínim orientativament. Per exemple, la xarxa d'aeroports d'USA, i la xarxa de practicants de *windsurf* presenten una modularitat força baixa, però tenen correlacions altes, i la xarxa de terroristes de l'11-S, o la d'emails, per una altra banda, tenen modularitat relativament altes però correlacions més aviat baixes. Així doncs, cal analitzar més en profunditat les xarxes tractades per comprovar què causa unes majors o menors correlacions.

Una altra observació general d'interès és que les correlacions per a les centralitats de vector propi i de subgraf són en general més baixes per a les xarxes considerades que les altres quatre. Si es comparen aquestes xarxes amb els grafs amb comunitats estàndard tractats es veu com, pel que fa a la distribució dels nodes a les comunitats, i el nombre de comunitats, en general s'assemblen més als grafs amb comunitats iguals. En quasi cap cas una sola comunitat acapara més d'un terç dels nodes, i quan existeix alguna comunitat més gran, n'hi han d'altres que tenen quasi la mateixa mida. Extrapolant les anàlisis realitzades amb aquests grafs aleatoris, això explicaria aquesta observació, i perquè les correlacions són quasi totes positives. L'única correlació negativa es dona per la centralitat de subgraf, que és la centralitat per a la que més fàcilment la correlació amb r és negativa, com s'ha vist, ja que és la que més dona valor a un entorn proper densament connectat del node; per a la xarxa d'aeroports d'Àsia, que és la que presenta una major diferència entre el percentatge de nodes a la comunitat més gran respecte la segona. Així doncs, sembla coincidir aquest resultat amb la interpretació donada, ja que totes les altres xarxes, menys la d'aeroports d'USA, tenen quantitats semblants de nodes a les dues, o fins i tot tres i quatre, comunitats més grans, per a la qual cosa es prioritza per a ser central respecte els fluxos a la xarxa exercir de frontera entre comunitats.

Es mostren a la figura 6 dues representacions gràfiques diferents de la xarxa d'aeroports asiàtics, on es poden observar les diferències comentades amb més profunditat. Sembla ser que la correlació negativa per a la centralitat de subgraf és deguda principalment a la comunitat representada de color verd, que tot i no ser la més gran, sinó la segona més gran, està molt més densament connectada que les altres. Així, els seus nodes, fins i tot els que tenen forces connexions amb altres comunitats, tenen una proporció d'arestes

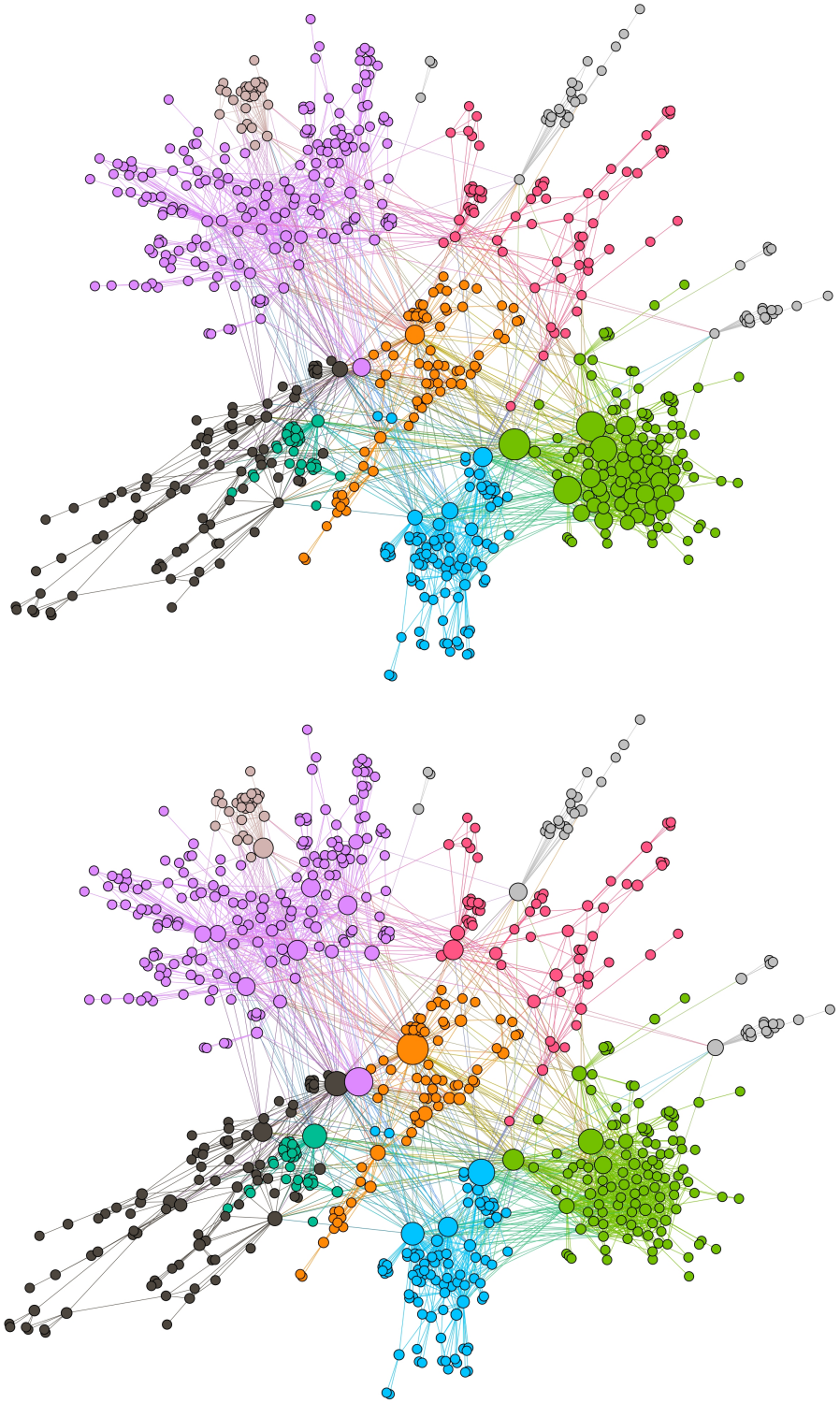


Figura 6: Xarxa d'aeroports d'Àsia, amb el color dels nodes representant la comunitat de pertinença, i la seva mida la centralitat de subgraf, a la gràfica superior, i la centralitat d'intermediació, a la inferior.

externes respecte les internes relativament baixa, i tenen valors per a aquesta centralitat clarament més alts que la resta de nodes del graf. Igual que passava amb els grafs amb comunitats estàndard amb comunitats diferents, no és tant el fet que els nodes amb una r més petita dins de cada comunitat siguin els més centrals que causa la correlació negativa, sinó que és la major densitat de connexions dins d'una comunitat.

En canvi, amb la centralitat d'intermediació cobren rellevància sobretot uns vèrtexs concrets, que són, com a mínim pel que fa a les comunitats més petites, els que les connecten amb la resta del graf. En el cas de la comunitat de color lila, la més gran, hi han alguns nodes més interns amb també centralitat d'intermediació força elevada. En aquest cas queda clar que, per a una xarxa d'aeroports, els nodes amb alta centralitat d'intermediació són els més importants en termes de comunicació a Àsia, i la centralitat de subgraf estaria més aviat relacionada amb la quantitat d'operacions de l'aeroport o similar.

Una altra xarxa que s'ha considerat útil representar és la dels terroristes de l'11-S, per a esbrinar perquè és en la que es correlacionen menys les diferents centralitat amb el ràtio r.

Figura 7: Xarxa de contactes entre els terroristes relacionats amb l'atentat de l'11-S a Madrid, on el color dels nodes representa la comunitat de pertinença, i la seva mida la centralitat de valor propi, a la gràfica superior, i la centralitat d'intermediació en flux, a la inferior.

Es veu doncs a la figura 12 com els nodes amb més centralitat d'intermediació en flux, degut a com estan estructurades les comunitats, no són tant els que les connecten entre elles, sinó més aviat els que connecten parts de la comunitat amb la resta. Un exemple clar són els nodes més centrals segons aquesta mesura en la comunitat verda, el més central a la taronja, i alguns de la lila. A més, cal tenir en compte el paper de l'algoritme escollit a l'hora de trobar els clústers, i els possibles errors en els que pot incórrer: per exemple, la comunitat de color verd fosc només disposa de tres nodes, i un d'ells té igual o mes connexions amb la comunitat verda, i amb la lila, que amb la seva. Així doncs, aquest node tindrà, amb aquesta assignació de comunitats, una proporció r molt alta, i tot i així no és central. Pel que fa a la centralitat de vector propi, veiem com en aquest cas la baixa correlació es deu a que les comunitats lila i verda estan força més densament connectades que la resta del graf, i aleshores els seus nodes que tenen grau més alt,

tot i ser intern a la comunitat, tenen centralitat de vector propi força alta.

Finalment, s'ha decidit representar també la xarxa de dofins, a la figura 8, per analitzar la alta correlació resultant entre la centralitat de proximitat i la r, respecte la centralitat de vector propi, per exemple.

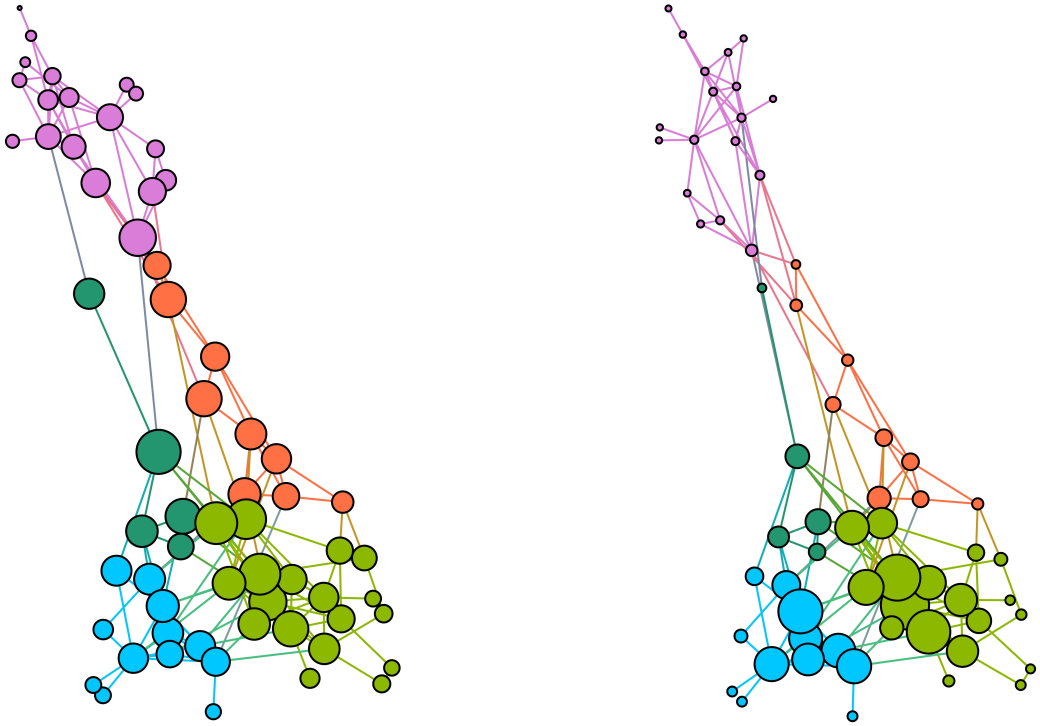


Figura 8: Xarxa de dofins amb el color dels nodes representant la comunitat de pertinença i la seva dimensió la centralitat de proximitat, a l'esquerra, i la de vector propi, a la dreta.

En aquest cas la forma allargada del graf, en el sentit de que la comunitat més gran, la de color lila, es troba separada de la verd clar i la blava, amb la verd fosc i la taronja exercint d'intermediaris, provoca la alta correlació amb la centralitat de proximitat. Així, els nodes que comuniquen entre clústers es troben situats més probablement al centre del graf, i per tant estan més a prop, en mitjana, de la resta de nodes del graf. És interessant destacar com aquesta interpretació s'aplica sobretot a les comunitats més externes, és a dir, la lila, la blava i la verd fosc, mentre que pel que fa a les altres dues no sembla que els nodes amb una r més gran siguin també els més centrals, pel que fa a la seva comunitat com a mínim. Si ens fixem ara en la centralitat de vector propi, veiem com prioritza els nodes situats a la part més densa en connexions, i amb més quantitat de nodes, del graf, a les comunitats verd clar i blau. Es veu doncs de nou el diferent punt de vista amb el que enfoquen la centralitat aquestes dues mesures, igual que a la figura 7; en un cas es dona prioritat als fluxos en la xarxa, i en l'altre als vèrtexs amb més connexions amb vèrtexs que també tenen moltes connexions, des d'un punt de vista més estructural, podríem dir.

4. Conclusions

Després de les anàlisis realitzades a les seccions 2 i 3, es poden extreure una sèrie de conclusions. Pel que fa a la secció 2, es conclou que, dels quatre algoritmes tractats, el Louvain és el que en mitjana és més precís a l'hora de detectar les comunitats dels grafs tractats, i a continuació vindrien el Girvan-Newman, l'Infomap, i el Miracom. Complementant els resultats obtinguts amb l'estudi de Lancichinetti i Fortunato [20], es poden extreure unes conclusions més avançades. Així, es conclou que el Louvain és l'algoritme més robust, en el sentit de que, sigui com sigui la xarxa a tractar, troba sempre una estructura de comunitats amb una exactitud mínimament bona. L'Infomap, per una altra banda, funciona millor com més gran sigui la xarxa, i a partir d'una certa dimensió és més precís que el Louvain. Això si, és un algoritme poc robust en el sentit de que o troba quasi exactament l'estructura de comunitats més adient per a la xarxa, o en determina una de molt incorrecta. Pel que fa al Girvan-Newman, és un algoritme molt precís quan la densitat d'arestes del graf és més aviat baixa, però és poc fiable en altres casos, com quan hi ha alguna comunitat molt gran, per exemple. El seu alt cost n'impossibilita l'ús per xarxes mitjanes, mentre que l'Infomap i el Louvain estan perfectament adaptats a xarxes amb més d'un milió de nodes [20]. L'algoritme Miracom, finalment, és clarament el que ha donat pitjors resultats. La conclusió general d'aquesta secció és doncs que, idealment, fer servir l'Infomap com el Louvain i comparar-ne els resultats permet trobar l'estructura en comunitats que més correspon a la xarxa en qüestió, i si és especialment poc densa i té pocs nodes resulta útil també fer servir el Girvan-Newman.

Passant ara a la secció 3, les conclusions soń les següents:

1. Per començar, la conclusió principal és que les correlacions entre r i les diferents mesures de centralitat són en general força altes, si les comunitats estan mínimament ben definides. Així, es conclou que el fet que un node estigui situat més o menys externament en la seva comunitat afecta a la seva centralitat, i que amb comunitats prou definides, és el factor més determinant si excloem el grau del node.
2. Per una altra banda, es conclou que aquestes correlacions seran totes positives sempre que no hi hagi alguna comunitat acaparant una proporció molt gran dels nodes del graf, o no excessivament gran però clarament més densa que la resta del graf. S'interpreta doncs que si les comunitats són relativament homogènies entre elles, i es reparteixen els vèrtexs del graf de forma a que no es concentren tots en una o unes poques, els nodes més importants de la xarxa seran els que exerceixen de frontera entre comunitats, comunicant entre elles. Cal però matisar aquestes afirmacions en funció d'altres característiques, més específiques, de cada xarxa.
3. Pel que fa a les diferents mesures de centralitat, s'extreuen les següents conclusions:
   (a) Les centralitats d'intermediació i d'intermediació en flux són les que tenen més tendència a tenir correlacions positives amb r (r és la proporció d'arestes externes a la comunitat respecte les internes d'un node), i, quan són negatives, amb la visualització de les xarxes s'ha comprovat que és degut a la major densitat d'arestes en una comunitat gran, i no a que els nodes estiguin a l'interior de les comunitats. Així, es conclou que, alhora d'exercir d'intermediari en els fluxos en una xarxa, els nodes més importants tendiran a estar a la frontera entre la seva comunitat i una altra.
   (b) La centralitat de subgraf és la que té més tendència a adoptar correlacions amb r negatives, i, quan es tracta de xarxes on totes les correlacions són positives, com es descriuen al punt 2, és la que presenta correlacions més petites. D'aquesta manera, com aquesta centralitat mesura la situació del node en un entorn densament connectat, considera més importants els nodes que es troben a l'interior de les comunitats si aquestes són prou denses, i hi ha poques connexions entre elles.

(c) La centralitat de vector propi es comporta de forma relativament similar a la centralitat de subgraf, però presenta generalment correlacions amb r més grans. Per exemple, en alguns casos la centralitat de subgraf té una correlació negativa, mentre que la de la centralitat de vector propi és pròxima al zero.

(d) Finalment, les centralitats de proximitat i de proximitat en flux presenten en general correlacions similars entre elles, i segueixen un patró proper a les dues centralitats basades en la intermediació. Tot i això, en el cas d'una comunitat bastant densa acaparant molts nodes, com per exemple quasi la meitat de la xarxa, adopta un comportament més proper a la centralitat de subgraf i de vector propi.

Es poden establir doncs dos tipus ideals de xarxa complexa: les que tenen comunitats similars, i sense cap excessivament gran, en les quals, des del punt de vista dels fluxos a la xarxa, els vèrtexs més importants són els que comuniquen entre comunitats; i els grafs amb una o poques comunitats molt grans, i/o més denses que les altres, en els quals els vèrtexs més centrals, des del punt de vista de l'estructura topològica local d'on es troba el vèrtex, estan a l'interior d'aquestes comunitats, mentre que exercir d'intermediari entre comunitats deixa de ser tant important. Quasi totes les xarxes extretes de dades empíriques tractades cauen en el primer tipus, i cal tenir en compte que es tracta només d'un model, ja que altres característiques de la xarxa poden jugar també un paper important, com s'ha comprovat amb l'anàlisi més específic de tres d'aquestes xarxes.

Com a possibles millores o investigacions futures relacionades amb el procediment seguit a la secció 3, es podria considerar utilitzar, enlloc del ràtio r, la diferència entre el valor de r de cada node i la mitja per a la seva comunitat, de manera a que fos realment una mesura permetent identificar els nodes més i menys externs a la seva comunitat, i no respecte el total del graf.

5. Referències

[1] Dolphins network dataset – konect, 2016. Disponible a `http://konect.uni-koblenz.de/networks/`, accedit: 14/6/2017.

[2] Train bombing network dataset – konect, 2016. Disponible a `http://konect.uni-koblenz.de/networks/`, accedit: 14/6/2017.

[3] Windsurfers network dataset – konect, 2016. Disponible a `http://konect.uni-koblenz.de/networks/`, accedit: 14/6/2017.

[4] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 10:10008, October 2008.

[5] U. Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30:136–145, 2008.

[6] U. Brandes and D. Fleischer. Centrality measures based on current flow. In *LCNS*, Proc. 22nd Symp. Theoretical Aspects of Computer Science (STACS '05), pages 533–544. Springer-Verlag, 2005.

[7] F. Comellas and A. Miralles. A fast and efficient algorithm to identify clusters in networks. *Applied Mathematics and Computation*, 217:2007–2014, November 2010.

[8] L. Danon, A. Díaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 9:09008, September 2005.

[9] E. Estrada. *The structure of complex networks: theory and applications*. Oxford University Press, 2011.

[10] E. Estrada and J.A. Rodriguez-Velazquez. Subgraf centrality in complex networks. *Physical Review, E.*, 71:056103, 2005.

[11] S. Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, February 2010.

[12] S. Fortunato and M. Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.

[13] L.C. Freeman. Centrality in networks: I. conceptual clarification. *Social Networks*, 1:215–239, 1979.

[14] P. Gleiser and L. Danon. Community Structure in Jazz. *Advances in Complex Systems*, 6:565–573, July 2003.

[15] B. H. Good, Y.-A. de Montjoye, and A. Clauset. Performance of modularity maximization in practical contexts. *Physical Review E.*, 81(4):046106, April 2010.

[16] C. Gros. *Complex and adaptative dynamical systems, a primer*. Springer, 2010.

[17] R. Guimerà, L. Danon, A. Díaz-Guilera, F. Giralt, and A. Arenas. Self-similar community structure in a network of human interactions. *Physical Review E*, 68(6):065103, December 2003.

[18] R. Guimerà, M. Sales-Pardo, and L. Amaral. Classes of complex networks defined by role-to-role connectivity profiles. *Nature physics*, 3:63–69, January 2007.

[19] F. Johnson Neil. *Simply complexity: A clear guide to complexity theory*. Oneworld publications, 2009.

[20] A. Lancichinetti and S. Fortunato. Community detection algorithms: a comparative analysis. *Physical Review E*, 80(5), nov 2009.

[21] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, oct 2008.

[22] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E.*, 69(2), February 2004.

[23] M.E.J Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27:39–54, 2005.

[24] M.E.J Newman. *Networks: an introduction*. Oxford University Press, 2010.

[25] M. Rovall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences (PNAS)*, 105:1118–1123, December 2007.

[26] A. Steger and N. C. Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8(4):377–396, July 1999.

[27] K. Stephenson and M. Zelen. Rethinking centrality: Methods and examples. *Social Networks*, 11:1–37, March 1989.

A. Annex: Programes en Python

A.1 Creació dels grafs aleatoris

En aquest apartat es presenten els codis utilitzats per generar els grafs amb comunitats estàndard i els grafs regulars amb comunitats.

Grafs amb comunitats estàndard

```
import networkx as nx

for i in xrange(0,25):
    Gi = nx.random_partition_graph([32,32,32,32],0.25,0.08)
    while (not nx.is_connected(Gi)):
        Gi = nx.random_partition_graph([32,32,32,32],0.25,0.08)
    nx.write_edgelist(Gi,'C:\Users\Alexi\Desktop\TFG\Grafs de prova\Amb networkx\Pin = 0,25 Pout = 0,08\comiguals
        ↪ \grafi025008'+str(i)+'.edgelist', data = False)

for i in xrange(0,25):
    Gi = nx.random_partition_graph([16,16,32,64],0.25,0.08)
    while (not nx.is_connected(Gi)):
        Gi = nx.random_partition_graph([16,16,32,64],0.25,0.08)
    nx.write_edgelist(Gi,'C:\Users\Alexi\Desktop\TFG\Grafs de prova\Amb networkx\Pin = 0,25 Pout = 0,08\
        ↪ comnoiguals\grafnoi025008'+str(i)+'.edgelist', data = False)

for i in xrange(0,25):
    Gi = nx.random_partition_graph([32,32,32,32],0.6,0.08)
    while (not nx.is_connected(Gi)):
        Gi = nx.random_partition_graph([32,32,32,32],0.6,0.08)
    nx.write_edgelist(Gi,'C:\Users\Alexi\Desktop\TFG\Grafs de prova\Amb networkx\Pin = 0,6 Pout = 0,08\comiguals\
        ↪ grafi06008'+str(i)+'.edgelist', data = False)

for i in xrange(0,25):
    Gi = nx.random_partition_graph([16,16,32,64],0.6,0.08)
    while (not nx.is_connected(Gi)):
        Gi = nx.random_partition_graph([16,16,32,64],0.6,0.08)
    nx.write_edgelist(Gi,'C:\Users\Alexi\Desktop\TFG\Grafs de prova\Amb networkx\Pin = 0,6 Pout = 0,08\
        ↪ comnoiguals\grafnoi06008'+str(i)+'.edgelist', data = False)

for i in xrange(0,25):
    Gi = nx.random_partition_graph([32,32,32,32],0.25,0.01)
    while (not nx.is_connected(Gi)):
        Gi = nx.random_partition_graph([32,32,32,32],0.25,0.01)
    nx.write_edgelist(Gi,'C:\Users\Alexi\Desktop\TFG\Grafs de prova\Amb networkx\Pin = 0,25 Pout = 0,01\comiguals
        ↪ \grafi025001'+str(i)+'.edgelist', data = False)

for i in xrange(0,25):
    Gi = nx.random_partition_graph([16,16,32,64],0.25,0.01)
    while (not nx.is_connected(Gi)):
        Gi = nx.random_partition_graph([16,16,32,64],0.25,0.01)
    nx.write_edgelist(Gi,'C:\Users\Alexi\Desktop\TFG\Grafs de prova\Amb networkx\Pin = 0,25 Pout = 0,01\
        ↪ comnoiguals\grafnoi025001'+str(i)+'.edgelist', data = False)
```

Grafs regulars amb comunitats

```
# -*- coding: utf-8 -*-
import networkx as nx
import random as rd
import numpy


#Creo grafs aleatoris regulars, amb 4 comunitats de 32 nodes cada una, amb pin sempre igual a 15/31 = 0,483 dos possibilitats per pout: 1/96 = 0,0104 o 8/96 = 0,0833
#Creo 4 grafs regulars de 32 nodes, i 32 conexions entre cada comunitat i les altres, una comunitat tindrà doncs (13*32)/2 - 32 = 176 conexions internes

for i in xrange(0,25):
    G1 = nx.random_regular_graph(15,32)
    G2 = nx.random_regular_graph(15,32)
    G3 = nx.random_regular_graph(15,32)
    G4 = nx.random_regular_graph(15,32)
    U1 = nx.disjoint_union(G1,G2)
    U2 = nx.disjoint_union(G3,G4)
    U = nx.disjoint_union(U1,U2)
    com = numpy.zeros((128,),dtype=numpy.int)
    for j in xrange(0,32):
        com[j] = 1
        com[j+32] = 2
        com[j+64] = 3
        com[j+96] = 4
    for j in xrange(0,16): #x = 32
        #primer desconecto un vèrtex aleatori de cada comunitat amb un dels seus veins, seleccionats aleatòriament, son els k1,l1, k2,l2, k3,l2 k4,l4
        k1 = rd.randint(0,31)
        ok = 0
        for a in xrange(0,len(U.neighbors(k1))):
            if com[U.neighbors(k1)[a]] == 1: #m'asseguro que el vèrtex seleccionat encara tingui veins a la comunitat a la que pertany
                ok = 1
        if ok == 0:
            k1 = rd.randint(0,31)
        veins = U.neighbors(k1)
        l = rd.randint(0,len(veins)-1)
        l1 = veins[l]
        while com[l1] != 1:
            l = rd.randint(0,len(veins)-1)
            l1 = veins[l]
        U.remove_edge(k1,l1)

        k2 = rd.randint(32,63)
        ok = 0
        for a in xrange(0,len(U.neighbors(k2))):
            if com[U.neighbors(k2)[a]] == 2:
                ok = 1
        if ok == 0:
            k2 = rd.randint(32,63)
        veins = U.neighbors(k2)
        l = rd.randint(0,len(veins)-1)
        l2 = veins[l]
```

```
while com[l2] != 2:
    l = rd.randint(0,len(veins)-1)
    l2 = veins[l]
U.remove_edge(k2,l2)

k3 = rd.randint(64,95)
ok = 0
for a in xrange(0,len(U.neighbors(k3))):
    if com[U.neighbors(k3)[a]] == 3:
        ok = 1
if ok == 0:
    k3 = rd.randint(64,95)
veins = U.neighbors(k3)
l = rd.randint(0,len(veins)-1)
l3 = veins[l]
while com[l3] != 3:
    l = rd.randint(0,len(veins)-1)
    l3 = veins[l]
U.remove_edge(k3,l3)

k4 = rd.randint(96,127)
ok = 0
for a in xrange(0,len(U.neighbors(k4))):
    if com[U.neighbors(k4)[a]] == 4:
        ok = 1
if ok == 0:
    k4 = rd.randint(96,127)
veins = U.neighbors(k4)
l = rd.randint(0,len(veins)-1)
l4 = veins[l]
while com[l4] != 4:
    l = rd.randint(0,len(veins)-1)
    l4 = veins[l]
U.remove_edge(k4,l4)
#ara uneixo aleatòriament els ki,li
pos = [k1,l1,k2,l2,k3,l3,k4,l4]
units = []
k = 0
while len(pos) > 0:
    k = k+1
    a = rd.choice(pos)
    pos.remove(a)
    b = rd.choice(pos)
    pos.append(a)
    if com[a] != com[b] and not U.has_edge(a,b):
        pos.remove(a)
        pos.remove(b)
        units.append([a,b])
    if (len(pos) == 2 and com[a] == com[b]) or k == 30:
        pos = [k1,l1,k2,l2,k3,l3,k4,l4]
        units = []
        k = 0

for k in xrange(0,len(units)):
    a = units[k][0]
    b = units[k][1]
    if (U.has_edge(a,b)):
```

```
                print 'fuck'
            U.add_edge(a,b)

    nx.write_edgelist(U,'/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_comunitats_regulars/pout_=_0,0104/grafreg001'
        ↪ +str(i)+'.edgelist', data = False)
print 'acabat'
for i in xrange(0,25):
    G1 = nx.random_regular_graph(15,32)
    G2 = nx.random_regular_graph(15,32)
    G3 = nx.random_regular_graph(15,32)
    G4 = nx.random_regular_graph(15,32)
    U1 = nx.disjoint_union(G1,G2)
    U2 = nx.disjoint_union(G3,G4)
    U = nx.disjoint_union(U1,U2)
    com = numpy.zeros((128,),dtype=numpy.int)
    for j in xrange(0,32):
        com[j] = 1
        com[j+32] = 2
        com[j+64] = 3
        com[j+96] = 4
    for j in xrange(0,112): #x = 32*7
        #primer desconecto un vèrtex aleatori de cada comunitat amb un dels seus veins, seleccionats aleatòriament, son
            ↪ els k1,l1, k2,l2, k3,l2 k4,l4

        ok = 0
        while ok == 0:
            k1 = rd.randint(0,31)
            for a in xrange(0,len(U.neighbors(k1))):
                if com[U.neighbors(k1)[a]] == 1: #m'asseguro que el vèrtex seleccionat encara tingui veins a la comunitat
                    ↪ a la que pertany
                    ok = 1
        veins = U.neighbors(k1)
        l = rd.randint(0,len(veins)-1)
        l1 = veins[l]
        while com[l1] != 1:
            l = rd.randint(0,len(veins)-1)
            l1 = veins[l]
        U.remove_edge(k1,l1)

        ok = 0
        while ok == 0:
            k2 = rd.randint(32,63)
            for a in xrange(0,len(U.neighbors(k2))):
                if com[U.neighbors(k2)[a]] == 2:
                    ok = 1
        veins = U.neighbors(k2)
        l = rd.randint(0,len(veins)-1)
        l2 = veins[l]
        while com[l2] != 2:
            l = rd.randint(0,len(veins)-1)
            l2 = veins[l]
        U.remove_edge(k2,l2)


        ok = 0
        while ok == 0:
            k3 = rd.randint(64,95)
```

```
        for a in xrange(0,len(U.neighbors(k3))):
            if com[U.neighbors(k3)[a]] == 3:
                ok = 1
    veins = U.neighbors(k3)
    l = rd.randint(0,len(veins)-1)
    l3 = veins[l]
    while com[l3] != 3:
        l = rd.randint(0,len(veins)-1)
        l3 = veins[l]
    U.remove_edge(k3,l3)

    ok = 0
    while ok == 0:
        k4 = rd.randint(96,127)
        for a in xrange(0,len(U.neighbors(k4))):
            if com[U.neighbors(k4)[a]] == 4:
                ok = 1
    veins = U.neighbors(k4)
    l = rd.randint(0,len(veins)-1)
    l4 = veins[l]
    while com[l4] != 4:
        l = rd.randint(0,len(veins)-1)
        l4 = veins[l]
    U.remove_edge(k4,l4)
    #ara uneixo aleatòriament els ki,li
    pos = [k1,l1,k2,l2,k3,l3,k4,l4]
    units = []
    k = 0
    while len(pos) > 0:
        k = k+1
        a = rd.choice(pos)
        pos.remove(a)
        b = rd.choice(pos)
        pos.append(a)
        if com[a] != com[b] and not U.has_edge(a,b):
            pos.remove(a)
            pos.remove(b)
            units.append([a,b])
        if (len(pos) == 2 and com[a] == com[b]) or k == 30:
            pos = [k1,l1,k2,l2,k3,l3,k4,l4]
            units = []
            k = 0
    for x in xrange(0,len(units)):
        U.add_edge(units[x][0],units[x][1])

nx.write_edgelist(U,'/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_comunitats_regulars/pout_=_0,1041/grafreg01'+
    ↪ str(i)+'.edgelist', data = False)
```

A.2 Algoritmes de detecció de comunitats

S'inclouen a continuació, com a funcions en Python, els codis utilitzats alhora d'aplicar cadascun dels algoritmes de detecció de comunitats als grafs. En el cas dels grafs LFR la funció per a l'algoritme Infomap és diferent, i és l'última que apareix.

```
import networkx as nx
```

```
import numpy
import itertools
import operator
import sys
sys.path.append('/home/aqm/Desktop/TFG/Infomap/examples/python')
import community
import math
from infomap import infomap
from optparse import OptionParser
try:
    import matplotlib.pyplot as plt
except:
    raise

from heapq import heappush, heappop
from operator import itemgetter

def qvalue(G, communities):
    """ Q value calculation
    This calculates the Q value according to "Finding and evaluating community
    structure in networks" by Newman and Girvan.

    @args
        G: graph
        communities: set of node sets denoting the communities

    @return
        Q value for the given community set
    """

    e = numpy.zeros( (len(communities), len(communities)))
    a = numpy.zeros( len(communities))
    com = {}

    numedges = G.number_of_edges()
    edgefrac = 1.0/(2*numedges)

    i = 0
    for c in communities:
        for n in c:
            com[n] = i

        i += 1

    totedges = 0
    for edge in G.edges_iter():
        i = com[edge[0]]
        j = com[edge[1]]

        e[i,j] += edgefrac
        e[j,i] += edgefrac
        a[i] += edgefrac
        a[j] += edgefrac
        totedges += 1
```

```
    return reduce(operator.add, (e[i,i] - a[i]**2 for i in xrange(len(a))))

# **********************************

def louvain(G):
    doResolution = 1.
    weighted = False
    if(len(sys.argv) > 2):
        doResolution = float(sys.argv[2])


    # do community detection and get dendrograph of communities
    dendo = community.generate_dendrogram(G, part_init=None, resolution=doResolution, weight='weight')

    # store communities at different levels
    parts = {}
    for level in range(0, len(dendo)):
        parts[level] = community.partition_at_level(dendo, level)

    # just do plain community detection instead of nested variant
    #levels = 1
    #parts[0] = community.best_partition(G) # find communities


    n = len(parts)-1
    com = numpy.zeros((len(parts[n]),),dtype=numpy.int)
    ncom = 0

    for i in xrange(0,len(parts[n])):
        com[i] = parts[n][i]
        if (parts[n][i] > ncom):
            ncom = parts[n][i]
    comunitats = []
    for i in xrange(0,ncom+1):
        comi = []
        for j in xrange(0,len(com)):
            if com[j] == i:
                comi.append(j)
        comunitats.append(comi)

    return comunitats


def girvannewman(g):

    def girvan_newman(G, most_valuable_edge=None):

        if G.number_of_edges() == 0:
            yield tuple(nx.connected_components(G))
            return
        # If no function is provided for computing the most valuable edge,
        # use the edge betweenness centrality.
        if most_valuable_edge is None:
            def most_valuable_edge(G):
                """Returns the edge with the highest betweenness centrality
                in the graph `G`.
```

```
            """
            # We have guaranteed that the graph is non-empty, so this
            # dictionary will never be empty.
            betweenness = nx.edge_betweenness_centrality(G)
            return max(betweenness, key=betweenness.get)
    # The copy of G here must include the edge weight data.
    g = G.copy().to_undirected()
    # Self-loops must be removed because their removal has no effect on
    # the connected components of the graph.
    g.remove_edges_from(g.selfloop_edges())
    while g.number_of_edges() > 0:
        yield _without_most_central_edges(g, most_valuable_edge)

def _without_most_central_edges(G, most_valuable_edge):
    """Returns the connected components of the graph that results from
    repeatedly removing the most "valuable" edge in the graph.

    'G' must be a non-empty graph. This function modifies the graph 'G'
    in-place; that is, it removes edges on the graph 'G'.

    'most_valuable_edge' is a function that takes the graph 'G' as input
    (or a subgraph with one or more edges of 'G' removed) and returns an
    edge. That edge will be removed and this process will be repeated
    until the number of connected components in the graph increases.

    """
    original_num_components = nx.number_connected_components(G)
    num_new_components = original_num_components
    while num_new_components <= original_num_components:
        edge = most_valuable_edge(G)
        G.remove_edge(*edge)
        new_components = tuple(nx.connected_components(G))
        num_new_components = len(new_components)
    return new_components

def qvalue(G, communities):
    """ Q value calculation
    This calculates the Q value according to "Finding and evaluating community
    structure in networks" by Newman and Girvan.

    @args
        G: graph
        communities: set of node sets denoting the communities

    @return
        Q value for the given community set
    """

    e = numpy.zeros( (len(communities), len(communities)))
    a = numpy.zeros( len(communities))
    com = {}

    numedges = G.number_of_edges()
    edgefrac = 1.0/(2*numedges)

    i = 0
    for c in communities:
```

```
            for n in c:
                com[n] = i

            i += 1

        totedges = 0
        for edge in G.edges_iter():
            i = com[edge[0]]
            j = com[edge[1]]

            e[i,j] += edgefrac
            e[j,i] += edgefrac
            a[i] += edgefrac
            a[j] += edgefrac
            totedges += 1

        return reduce(operator.add, (e[i,i] - a[i]**2 for i in xrange(len(a))))

    it = girvan_newman(g)
    A=list(it)

    mod = []
    for k in xrange(0,len(A)):
        mod.append(qvalue(g,A[k]))

    maxi = 0
    i = 0
    for k in xrange(0,len(mod)):
        if mod[k] > maxi:
            maxi = mod[k]
            i = k
    com = []
    for j in xrange(0,len(A[i])):
        comi = []
        for x in A[i][j]:
            comi.append(x)
        com.append(comi)
    return com

def miracom(G):

    def elimina_L(x,usat,L):
        #Funció que busca el vèrtex x a L, el marca com a usat, i l'elimina de L i L_dic
        usat[x] = 1
        L.remove(x)


    def n_avgdeg(C):
        Caux = []
        enter = 0
        for i in xrange(0,len(C)):
            if type(C[i]) is int:
                enter = 1
                break
            for j in xrange (0,len(C[i])):
                Caux.append(C[i][j])
        if enter == 0:
```

```
        C = Caux
    suma = 0
    nodes = numpy.zeros((n,),dtype=numpy.int)
    for i in xrange(0,len(C)):
        nodes[C[i]] = 1
    for i in xrange(0,len(C)):
        dins = 0

        veins = list(nx.all_neighbors(G,C[i]))
        for j in xrange(0,len(veins)):
            if nodes[veins[j]] == 1:
                dins = dins+1
        if dins > 0:
            suma = suma + dins/len(veins)
    return suma/len(C)

#parser = argparse.ArgumentParser()
#parser.add_argument('-o', action='store', dest='o', type=str, required = True)
#inargs = parser.parse_args()
#G = nx.parse_edgelist(inargs.o)


#ordenem la llista de vèrtexs en funció del seu grau
L_dic = sorted(G.degree_iter(),key=itemgetter(1),reverse=True)
L = [i[0] for i in L_dic]

n = len(L) # n és el nombre total de vèrtexs al graf
Lpre = []
usat = numpy.zeros((n,),dtype=numpy.int)
k = 0 #nombre de comunitats
comunitats = []
suma = 0
for i in xrange(0,n):
    suma = suma + nx.degree(G,i)
graumig = suma/n

while len(L) > 0:
    Cpre = []
    f = 0
    #seleccionem el vertex inicial per a formar una nova comunitat
    if k == 0:
        f = L[0]
    else:
        f = L[0]
        for i in xrange(0,len(L)):
            suma = 0
            veins = list(nx.all_neighbors(G,L[i]))
            for j in xrange(0,len(veins)):
                if usat[veins[j]] == 0: #contem els veins que encara no hem fet servir
                    suma = suma+1
            if suma*2 >= len(veins):
                f = L[i]
                break

    #Creem L1 amb f i els vèrtexs a distància 1 en L i L2 amb els a distància 2 també en L
    veins = list(nx.all_neighbors(G,f))
    L1 = [f]
```

```
L2 = []
dins_L1 = numpy.zeros((n),dtype=numpy.int)
dins_L2 = numpy.zeros((n),dtype=numpy.int)
dins_L1[f] = 1
for i in xrange(0,len(veins)):

    if usat[veins[i]] == 0: #L1
        L1.append(veins[i])
        dins_L1[veins[i]] = 1

for i in xrange(0,len(L1)):
    veins_i = list(nx.all_neighbors(G,L1[i]))
    for j in xrange(0,len(veins_i)):
        if usat[veins_i[j]] == 0 and dins_L1[veins_i[j]] == 0 and dins_L2[veins_i[j]] == 0: #L2
            L2.append(veins_i[j])
            dins_L2[veins_i[j]] = 1

#Ara seleccionem en L1 els vèrtexs per a la nostra comunitat

for i in xrange(0,len(L1)):
    veins_i = list(nx.all_neighbors(G,L1[i]))
    l = 0
    m = 0
    for j in xrange(0,len(veins_i)):
        if dins_L1[veins_i[j]] == 1:
            l=l+1 #Contem les adjacències en L1
        if dins_L2[veins_i[j]] == 1:
            m=m+1 #Contem les adjacències en L2

    if l >= m and (l+m)*2 >= nx.degree(G,L1[i]):
        Cpre.append(L1[i])


if len(Cpre) == 0: #mirem si hem acabat de formar comunitats
    break

#Ara anexem o no Cpre a una comunitat ja existent

if k == 0 and len(Cpre) > 2:
    comunitats.append(Cpre)
    for i in xrange(0,len(Cpre)):
        elimina_L(Cpre[i],usat,L)
    k=k+1
else:
    trobat = 0
    for i in xrange(0,k):
        enC = numpy.zeros((n,),dtype=numpy.int)
        for j in xrange(0,len(comunitats[i])): #Marquem els vèrtexs de la comunitat ja formada
            enC[comunitats[i][j]] = 1
        suma = 0
        for j in xrange(0,len(Cpre)):
            veins = list(nx.all_neighbors(G,Cpre[j]))
            for l in xrange(0,len(veins)):
                if enC[veins[l]] == 1: #contem les adjacències entre les dues comunitats
                    suma = suma+1

        if n_avgdeg([Cpre,comunitats[i]]) >= n_avgdeg(Cpre) and n_avgdeg([Cpre,comunitats[i]]) >= n_avgdeg(
```

```
                    ↪ comunitats[i]) and suma > len(Cpre)*len(comunitats[i])*graumig/(n-1):
                trobat = 1
                for j in xrange(0,len(Cpre)):
                    comunitats[i].append(Cpre[j])
                    elimina_L(Cpre[j],usat,L)
                break
        if trobat == 0:
            if len(Cpre) > 2:
                comunitats.append(Cpre)
                for i in xrange(0,len(Cpre)):
                    elimina_L(Cpre[i],usat,L)
                k = k+1
            else:
                for i in xrange(0,len(Cpre)):
                    Lpre.append(Cpre[i])
                    elimina_L(Cpre[i],usat,L)

    #Ja em format totes les comunitats possibles, ara afegim els nodes restants a la comunitat amb la que tenen més adjac
        ↪ ències

    com = numpy.zeros((n,),dtype=numpy.int)
    for i in xrange(0,len(com)):
        com[i] = com[i]-1
    for i in xrange(0,len(comunitats)): #Creem un vector com que per a cada node té la comunitat al qual pertany
        for j in xrange(0,len(comunitats[i])):
            com[comunitats[i][j]] = i

    if (len(Lpre)>0):
        for i in xrange(0,len(Lpre)):
            L.append(Lpre[i])
    if (len(comunitats) == 0):
        return G.nodes()
    if (len(L) > 0):
        for i in xrange(0,len(L)):
            veins = list(nx.all_neighbors(G,L[i]))
            adj = numpy.zeros((len(comunitats),),dtype=numpy.int)
            for j in xrange(0,len(veins)):
                if com[veins[j]] != -1:
                    adj[com[veins[j]]] = adj[com[veins[j]]] + 1

            maxpos = numpy.argmax(adj)
            comunitats[maxpos].append(L[i])

    return comunitats

def Imap(G):
    infomapW = infomap.Infomap("-z -u -i link-list --silent")
    for e in G.edges_iter():
        infomapW.addLink(*e)
    infomapW.run()
    tree = infomapW.tree
    n = tree.numTopModules()
    com = [[] for i in range(n)]
    for node in tree.leafIter():
        com[node.moduleIndex()].append(node.physIndex)
    return com
```

```
def lmap(j,i):
    infomapW = infomap.Infomap("-u -i 'link-list' --silent")
    if j == 1:
        infomapW.readInputData('/home/aqm/Desktop/TFG/Grafs de prova/Amb LFR benchmark/mu = 0,2/graf'+str(
            ↪ i)+'/network.dat')
    if j == 2:
        infomapW.readInputData('/home/aqm/Desktop/TFG/Grafs de prova/Amb LFR benchmark/mu = 0,5/graf'+str(
            ↪ i)+'/network.dat')
    infomapW.run()
    tree = infomapW.tree
    n = tree.numTopModules()
    com = [[] for i in range(n)]
    for node in tree.leafIter():
        com[node.moduleIndex()].append(node.physIndex+1)
    return com
```

A.3 Comparació de la precisió dels diferents algoritmes

S'inclou en aquest apartat la implementació de la informació mútua normalitzada per a cada tipus de graf, tenint en compte que en els grafs LFR l'estructura de comunitats ve donada en un arxiu community.dat per a cada graf, i el codi destinat al càlcul de les mitjanes d'aquesta quantitat a continuació.

Grafs amb comunitats estàndard

```
def norminformationI(com):
    #creem la matriu de confusió
    if len(com) == 128:
        return 0
    if len(com) == 1:
        return 0
    conf = [[0] * len(com) for x in range(4)]
    for i in xrange(0,len(com)):
        for j in xrange(0,len(com[i])):
            if com[i][j] < 32:
                conf[0][i] = conf[0][i] +1
            if com[i][j] >= 32 and com[i][j] < 64:
                conf[1][i] = conf[1][i] + 1
            if com[i][j] >= 64 and com[i][j] < 96:
                conf[2][i] = conf[2][i] + 1
            if com[i][j] >= 96 and com[i][j] < 128:
                conf[3][i] = conf[3][i] + 1
    #calculem l'index normalitzat d'informació mútua
    colconf = numpy.zeros((len(com),),dtype=numpy.int)
    rowconf = numpy.zeros(4,dtype=numpy.int)
    for i in xrange(0,4):
        for j in xrange(0,len(com)):
            colconf[j] = colconf[j] + conf[i][j]
            rowconf[i] = rowconf[i] + conf[i][j]
    I = 0
    for i in xrange(0,4):
        for j in xrange(0,len(com)):
            if conf[i][j] != 0:
                I = I + conf[i][j]*math.log(float((float(conf[i][j])*128))/float((rowconf[i]*colconf[j])))
```

```
    I = -2*I
    H1 = 0
    for i in xrange(0,4):
        if rowconf[i] != 0:
            H1 = H1 + rowconf[i]*math.log(float(float(rowconf[i])/128))
    H2 = 0
    for j in xrange(0,len(com)):
        if colconf[j] != 0:
            H2 = H2 + colconf[j]*math.log(float(float(colconf[j])/128))
    return I/(H2+H1)

def norminformationNOI(com):
    if len(com) == 128:
        return 0
    if len(com) == 1:
        return 0
    #creem la matriu de confusió
    conf = [[0] * len(com) for x in range(4)]
    for i in xrange(0,len(com)):
        for j in xrange(0,len(com[i])):
            if com[i][j] < 16:
                conf[0][i] = conf[0][i] +1
            if com[i][j] >= 16 and com[i][j] < 32:
                conf[1][i] = conf[1][i] + 1
            if com[i][j] >= 32 and com[i][j] < 64:
                conf[2][i] = conf[2][i] + 1
            if com[i][j] >= 64 and com[i][j] < 128:
                conf[3][i] = conf[3][i] + 1

    #calculem l'index normalitzat d'informació mútua
    colconf = numpy.zeros((len(com),),dtype=numpy.int)
    rowconf = numpy.zeros(4,dtype=numpy.int)
    for i in xrange(0,4):
        for j in xrange(0,len(com)):
            colconf[j] = colconf[j] + conf[i][j]
            rowconf[i] = rowconf[i] + conf[i][j]
    I = 0
    for i in xrange(0,4):
        for j in xrange(0,len(com)):
            if conf[i][j] != 0:
                I = I + conf[i][j]*math.log(float((float(conf[i][j])*128))/float((rowconf[i]*colconf[j])))
    I = -2*I
    H1 = 0
    for i in xrange(0,4):
        if rowconf[i] != 0:
            H1 = H1 + rowconf[i]*math.log(float(float(rowconf[i])/128))
    H2 = 0
    for j in xrange(0,len(com)):
        if colconf[j] != 0:
            H2 = H2 + colconf[j]*math.log(float(float(colconf[j])/128))
    return I/(H2+H1)

# guardo les mitjes norminf per als 25 grafs de cada tipus en un vector on pos 0:0,25-0,08 igu, 1:0,25-0,08 dif,
    ↪ 2:0,6-0,08 igu, 3:0,6-0,08 dif, 4:0,25-0,01 igu, 5:0,25-0,01 dif
louv = []
mira = []
girvnewm = []
```

```
infomapa = []

# mitja és un vector on a cada posició sumo els norminf dels 25 grafs per a cada algoritme, 0:louvain 1:Miracom 2:
    ↪ girvannewman 3: clausetnewman
mitja = numpy.zeros((4,),dtype=numpy.float)
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_networkx/Pin_=_0,25_Pout_=_0,08/comiguals/
        ↪ grafi025008'+str(i)+'.edgelist',nodetype = int, data = False)
    com = louvain(G)
    mitja[0] = mitja[0] + norminformationI(com)
    com = miracom(G)
    mitja[1] = mitja[1] + norminformationI(com)
    com = girvannewman(G)
    mitja[2] = mitja[2] + norminformationI(com)
    com = Imap(G)
    mitja[3] = mitja[3] + norminformationI(com)
louv.append(mitja[0]/25)
mira.append(mitja[1]/25)
girvnewm.append(mitja[2]/25)
infomapa.append(mitja[3]/25)
mitja = numpy.zeros((4,),dtype=numpy.float)
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_networkx/Pin_=_0,25_Pout_=_0,08/
        ↪ comnoiguals/grafnoi025008'+str(i)+'.edgelist',nodetype = int, data = False)
    com = louvain(G)
    mitja[0] = mitja[0] + norminformationNOI(com)
    com = miracom(G)
    mitja[1] = mitja[1] + norminformationNOI(com)
    com = girvannewman(G)
    mitja[2] = mitja[2] + norminformationNOI(com)
    com = Imap(G)
    mitja[3] = mitja[3] + norminformationNOI(com)
louv.append(mitja[0]/25)
mira.append(mitja[1]/25)
girvnewm.append(mitja[2]/25)
infomapa.append(mitja[3]/25)
mitja = numpy.zeros((4,),dtype=numpy.float)
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_networkx/Pin_=_0,6_Pout_=_0,08/comiguals/
        ↪ grafi06008'+str(i)+'.edgelist',nodetype = int, data = False)
    com = louvain(G)
    mitja[0] = mitja[0] + norminformationI(com)
    com = miracom(G)
    mitja[1] = mitja[1] + norminformationI(com)
    com = girvannewman(G)
    mitja[2] = mitja[2] + norminformationI(com)
    com = Imap(G)
    mitja[3] = mitja[3] + norminformationI(com)
louv.append(mitja[0]/25)
mira.append(mitja[1]/25)
girvnewm.append(mitja[2]/25)
infomapa.append(mitja[3]/25)
mitja = numpy.zeros((4,),dtype=numpy.float)
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_networkx/Pin_=_0,6_Pout_=_0,08/comnoiguals
        ↪ /grafnoi06008'+str(i)+'.edgelist',nodetype = int, data = False)
    com = louvain(G)
```

```
    mitja[0] = mitja[0] + norminformationNOI(com)
    com = miracom(G)
    mitja[1] = mitja[1] + norminformationNOI(com)
    com = girvannewman(G)
    mitja[2] = mitja[2] + norminformationNOI(com)
    com = Imap(G)
    mitja[3] = mitja[3] + norminformationNOI(com)
louv.append(mitja[0]/25)
mira.append(mitja[1]/25)
girvnewm.append(mitja[2]/25)
infomapa.append(mitja[3]/25)
mitja = numpy.zeros((4,),dtype=numpy.float)
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_networkx/Pin_=_0,25_Pout_=_0,01/comiguals/
        ↪ grafi025001'+str(i)+'.edgelist', nodetype = int, data = False)
    com = louvain(G)
    mitja[0] = mitja[0] + norminformationI(com)
    com = miracom(G)
    mitja[1] = mitja[1] + norminformationI(com)
    com = girvannewman(G)
    mitja[2] = mitja[2] + norminformationI(com)
    com = Imap(G)
    mitja[3] = mitja[3] + norminformationI(com)
louv.append(mitja[0]/25)
mira.append(mitja[1]/25)
girvnewm.append(mitja[2]/25)
infomapa.append(mitja[3]/25)
mitja = numpy.zeros((4,),dtype=numpy.float)
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_networkx/Pin_=_0,25_Pout_=_0,01/
        ↪ comnoiguals/grafnoi025001'+str(i)+'.edgelist',nodetype = int, data = False)
    com = louvain(G)
    mitja[0] = mitja[0] + norminformationNOI(com)
    com = miracom(G)
    mitja[1] = mitja[1] + norminformationNOI(com)
    com = girvannewman(G)
    mitja[2] = mitja[2] + norminformationNOI(com)
    com = Imap(G)
    mitja[3] = mitja[3] + norminformationNOI(com)
louv.append(mitja[0]/25)
mira.append(mitja[1]/25)
girvnewm.append(mitja[2]/25)
infomapa.append(mitja[3]/25)

print 'Les_mitjes_de_la_mutual_normalized_information_per_louvain_són', louv
print 'Les_mitjes_de_la_mutual_normalized_information_per_miracom_són', mira
print 'Les_mitjes_de_la_mutual_normalized_information_per_girvan-newman_són', girvnewm
print 'Les_mitjes_de_la_mutual_normalized_information_per_infomap_són', infomapa
```

Grafs LFR

```
def norminformationI(comfa,comre):
    #creem la matriu de confusió
    if len(comfa) == 128:
```

```
        return 0
    comfavec = numpy.zeros((129,),dtype = numpy.int)
    for i in xrange (0,len(comfa)):
        for j in xrange(0, len(comfa[i])):
            comfavec[comfa[i][j]] = i

    conf = [[0] * len(comfa) for x in range(len(comre))]
    for i in xrange(0,len(comre)):
        for j in xrange(0,len(comre[i])):
            conf[i][comfavec[comre[i][j]]] = conf[i][comfavec[comre[i][j]]] + 1

    #calculem l'index normalitzat d'informació mútua
    colconf = numpy.zeros((len(comfa),),dtype=numpy.int)
    rowconf = numpy.zeros(len(comre),dtype=numpy.int)
    for i in xrange(0,len(comre)):
        for j in xrange(0,len(comfa)):
            colconf[j] = colconf[j] + conf[i][j]
            rowconf[i] = rowconf[i] + conf[i][j]
    I = 0
    for i in xrange(0,len(comre)):
        for j in xrange(0,len(comfa)):
            if conf[i][j] != 0:
                I = I + conf[i][j]*math.log(float((float(conf[i][j])*128))/float((rowconf[i]*colconf[j])))
    I = -2*I
    H1 = 0
    for i in xrange(0,len(comre)):
        if rowconf[i] != 0:
            H1 = H1 + rowconf[i]*math.log(float(float(rowconf[i])/128))
    H2 = 0
    for j in xrange(0,len(comfa)):
        if colconf[j] != 0:
            H2 = H2 + colconf[j]*math.log(float(float(colconf[j])/128))
    return I/(H2+H1)


# guardo les mitjes norminf per als 25 grafs de cada tipus en un vector on pos 0:pout = 0.0104 1:pout = 0.104
louv = []
mira = []
girvnewm = []
infomapa = []

# mitja és un vector on a cada posició sumo els norminf dels 25 grafs per a cada algoritme, 0:louvain 1:Miracom 2:
    ↪ girvannewman 3: clausetnewman
mitja = numpy.zeros((4,),dtype=numpy.float)
for i in xrange(1,26):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_LFR_benchmark/mu_=_0,2/graf'+str(i)+'/
        ↪ network.dat',nodetype = int, data = False)
    s = open('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_LFR_benchmark/mu_=_0,2/graf'+str(i)+'/community.dat
        ↪ ', 'r')
    veccom = numpy.zeros((129,),dtype=numpy.int)
    maxi = 0
    for line in s:
        a = line.split()
        a = [int(x) for x in a]
        veccom[a[0]] = a[1]
        if a[1] > maxi:
            maxi = a[1]
```

```
    comre = [[] * 1 for k in range(maxi)]
    for j in xrange(1,129):
        comre[veccom[j]-1].append(j)

    comfa = louvain(G)
    mitja[0] = mitja[0] + norminformationl(comfa,comre)
    comfa = miracom(G)
    mitja[1] = mitja[1] + norminformationl(comfa,comre)
    comfa = girvannewman(G)
    mitja[2] = mitja[2] + norminformationl(comfa,comre)
    comfa = lmap(1,i)
    mitja[3] = mitja[3] + norminformationl(comfa,comre)
louv.append(mitja[0]/25)
mira.append(mitja[1]/25)
girvnewm.append(mitja[2]/25)
infomapa.append(mitja[3]/25)
mitja = numpy.zeros((4,),dtype=numpy.float)

for i in xrange(1,26):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_LFR_benchmark/mu_=_0,5/graf'+str(i)+'/
        ↪ network.dat',nodetype = int, data = False)
    s = open('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_LFR_benchmark/mu_=_0,5/graf'+str(i)+'/community.dat
        ↪ ', 'r')
    veccom = numpy.zeros((129,),dtype=numpy.int)
    maxi = 0
    for line in s:
        a = line.split()
        a = [int(x) for x in a]
        veccom[a[0]] = a[1]
        if a[1] > maxi:
            maxi = a[1]
    comre = [[] * 1 for k in range(maxi)]
    for j in xrange(1,129):
        comre[veccom[j]-1].append(j)

    comfa = louvain(G)
    mitja[0] = mitja[0] + norminformationl(comfa,comre)
    comfa = miracom(G)
    mitja[1] = mitja[1] + norminformationl(comfa,comre)
    comfa = girvannewman(G)
    mitja[2] = mitja[2] + norminformationl(comfa,comre)
    comfa = lmap(2,i)
    mitja[3] = mitja[3] + norminformationl(comfa,comre)
louv.append(mitja[0]/25)
mira.append(mitja[1]/25)
girvnewm.append(mitja[2]/25)
infomapa.append(mitja[3]/25)
mitja = numpy.zeros((4,),dtype=numpy.float)

print 'Les_mitjes_de_la_mutual_normalized_information_per_louvain_són', louv
print 'Les_mitjes_de_la_mutual_normalized_information_per_miracom_són', mira
print 'Les_mitjes_de_la_mutual_normalized_information_per_girvan-newman_són', girvnewm
print 'Les_mitjes_de_la_mutual_normalized_information_per_infomap_són', infomapa
```

Grafs regulars amb comunitats

```
def norminformationI(com):
    #creem la matriu de confusió
    if len(com) == 128:
        return 0
    if len(com) == 1:
        return 0
    conf = [[0] * len(com) for x in range(4)]
    for i in xrange(0,len(com)):
        for j in xrange(0,len(com[i])):
            if com[i][j] < 32:
                conf[0][i] = conf[0][i] +1
            if com[i][j] >= 32 and com[i][j] < 64:
                conf[1][i] = conf[1][i] + 1
            if com[i][j] >= 64 and com[i][j] < 96:
                conf[2][i] = conf[2][i] + 1
            if com[i][j] >= 96 and com[i][j] < 128:
                conf[3][i] = conf[3][i] + 1
    #calculem l'index normalitzat d'informació mútua
    colconf = numpy.zeros((len(com),),dtype=numpy.int)
    rowconf = numpy.zeros(4,dtype=numpy.int)
    for i in xrange(0,4):
        for j in xrange(0,len(com)):
            colconf[j] = colconf[j] + conf[i][j]
            rowconf[i] = rowconf[i] + conf[i][j]
    I = 0
    for i in xrange(0,4):
        for j in xrange(0,len(com)):
            if conf[i][j] != 0:
                I = I + conf[i][j]*math.log(float((float(conf[i][j])*128))/float((rowconf[i]*colconf[j])))
    I = -2*I
    H1 = 0
    for i in xrange(0,4):
        if rowconf[i] != 0:
            H1 = H1 + rowconf[i]*math.log(float(float(rowconf[i])/128))
    H2 = 0
    for j in xrange(0,len(com)):
        if colconf[j] != 0:
            H2 = H2 + colconf[j]*math.log(float(float(colconf[j])/128))
    return I/(H2+H1)


# guardo les mitjes norminf per als 25 grafs de cada tipus en un vector on pos 0:pout = 0.0104 1:pout = 0.104
louv = []
mira = []
girvnewm = []
infomapa = []

# mitja és un vector on a cada posició sumo els norminf dels 25 grafs per a cada algoritme, 0:louvain 1:Miracom 2:
    ↪ girvannewman 3: clausetnewman
mitja = numpy.zeros((4,),dtype=numpy.float)
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_comunitats_regulars/pout_=_0,0104/grafreg001
        ↪ '+str(i)+'.edgelist',nodetype = int, data = False)
    com = louvain(G)
    mitja[0] = mitja[0] + norminformationI(com)
```

```
    com = miracom(G)
    mitja[1] = mitja[1] + norminformationI(com)
    com = girvannewman(G)
    mitja[2] = mitja[2] + norminformationI(com)
    com = Imap(G)
    mitja[3] = mitja[3] + norminformationI(com)
louv.append(mitja[0]/25)
mira.append(mitja[1]/25)
girvnewm.append(mitja[2]/25)
infomapa.append(mitja[3]/25)
mitja = numpy.zeros((4,),dtype=numpy.float)

for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_comunitats_regulars/pout_=_0,1041/grafreg01'
        ↪ +str(i)+'.edgelist',nodetype = int, data = False)
    com = louvain(G)
    mitja[0] = mitja[0] + norminformationI(com)
    com = miracom(G)
    mitja[1] = mitja[1] + norminformationI(com)
    com = girvannewman(G)
    mitja[2] = mitja[2] + norminformationI(com)
    com = Imap(G)
    mitja[3] = mitja[3] + norminformationI(com)
louv.append(mitja[0]/25)
mira.append(mitja[1]/25)
girvnewm.append(mitja[2]/25)
infomapa.append(mitja[3]/25)
mitja = numpy.zeros((4,),dtype=numpy.float)

print 'Les_mitjes_de_la_mutual_normalized_information_per_louvain_són', louv
print 'Les_mitjes_de_la_mutual_normalized_information_per_miracom_són', mira
print 'Les_mitjes_de_la_mutual_normalized_information_per_girvan−newman_són', girvnewm
print 'Les_mitjes_de_la_mutual_normalized_information_per_infomap_són', infomapa
```

A.4 Programes de la secció 3

S'inclouen a continuació els codis utilitzats a la secció 3 del treball. Primer s'inclouen les funcions generals a totes les xarxes utilitzades, és a dir, el càlcul de r, i de la correlació entre dues llistes, i després els codis per a cada cas.

Càlcul de la proporció r

```
def outin(com,v,G):
    #calculates the proportion of links out/in of the community of node v, given communities com and graph G
    n = len(G)
    veccom = numpy.zeros((n,), dtype=numpy.int)
    for i in xrange(0,len(com)):
        for j in xrange(0,len(com[i])):
            veccom[com[i][j]] = i

    veins = G.neighbors(v)
    internal = 0.0
```

```
    external = 0.0
    for i in xrange(0,len(veins)):
        if veccom[veins[i]] == veccom[v]:
            internal = internal + 1
        if veccom[veins[i]] != veccom[v]:
            external = external + 1
    if internal == 0:
        return float(float(external)/0.1)
    return float(float(external)/float(internal))
```

[language = Python]

Funció de càlcul de correlacions

```
def average(x):
    if type(x) == list:
        return float(sum(x)) / len(x)
    if type(x) == dict:
        return float(sum(x.values())) / len(x)

def pearson_def(x, y):
    assert len(x) == len(y)
    n = len(x)
    assert n > 0
    avg_x = average(x)
    avg_y = average(y)
    diffprod = 0
    xdiff2 = 0
    ydiff2 = 0
    for idx in range(n):
        xdiff = x[idx] - avg_x
        ydiff = y[idx] - avg_y
        diffprod += xdiff * ydiff
        xdiff2 += xdiff * xdiff
        ydiff2 += ydiff * ydiff

    return diffprod / math.sqrt(xdiff2 * ydiff2)
```

Grafs amb comunitats estàndard

```
mitja025008i = {'bet_cen':0,'com_cen':0, 'clo_cen':0, 'eig_cen':0, 'pg_rk':0, 'fl_clo_cen':0, 'fl_bet_cen':0}
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_networkx/Pin_=_0,25_Pout_=_0,08/comiguals/
        ↪ grafi025008'+str(i)+'.edgelist',nodetype = int, data = False)
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
    pg_rk = nx.pagerank(G)
    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    com = [[] for x in range(4)]
    for i in xrange(0,32):
```

```
            com[0].append(i)
            com[1].append(i+32)
            com[2].append(i+64)
            com[3].append(i+96)
        l = []
        for v in nx.nodes(G):
            l.append(outin(com,v,G))
        mitja025008i['bet_cen'] = mitja025008i['bet_cen'] + pearson_def(l,bet_cen)
        mitja025008i['com_cen'] = mitja025008i['com_cen'] + pearson_def(l,com_cen)
        mitja025008i['clo_cen'] = mitja025008i['clo_cen'] + pearson_def(l,clo_cen)
        mitja025008i['eig_cen'] = mitja025008i['eig_cen'] + pearson_def(l,eig_cen)
        mitja025008i['pg_rk'] = mitja025008i['pg_rk'] + pearson_def(l,pg_rk)
        mitja025008i['fl_clo_cen'] = mitja025008i['fl_clo_cen'] + pearson_def(l,fl_clo_cen)
        mitja025008i['fl_bet_cen'] = mitja025008i['fl_bet_cen'] + pearson_def(l,fl_bet_cen)

for key in mitja025008i:
    mitja025008i[key] = mitja025008i[key]/25

mitja025008noi = {'bet_cen':0,'com_cen':0, 'clo_cen':0, 'eig_cen':0, 'pg_rk':0, 'fl_clo_cen':0, 'fl_bet_cen':0}
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_networkx/Pin_=_0,25_Pout_=_0,08/
        ↪ comnoiguals/grafnoi025008'+str(i)+'.edgelist',nodetype = int, data = False)
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
    pg_rk = nx.pagerank(G)
    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    com = [[] for x in range(4)]
    for i in xrange(0,16):
        com[0].append(i)
        com[1].append(i+16)
        com[2].append(i+32)
        com[2].append(i+48)
        com[3].append(i+64)
        com[3].append(i+80)
        com[3].append(i+96)
        com[3].append(i+112)
    l = []
    for v in nx.nodes(G):
        l.append(outin(com,v,G))
    mitja025008noi['bet_cen'] = mitja025008noi['bet_cen'] + pearson_def(l,bet_cen)
    mitja025008noi['com_cen'] = mitja025008noi['com_cen'] + pearson_def(l,com_cen)
    mitja025008noi['clo_cen'] = mitja025008noi['clo_cen'] + pearson_def(l,clo_cen)
    mitja025008noi['eig_cen'] = mitja025008noi['eig_cen'] + pearson_def(l,eig_cen)
    mitja025008noi['pg_rk'] = mitja025008noi['pg_rk'] + pearson_def(l,pg_rk)
    mitja025008noi['fl_clo_cen'] = mitja025008noi['fl_clo_cen'] + pearson_def(l,fl_clo_cen)
    mitja025008noi['fl_bet_cen'] = mitja025008noi['fl_bet_cen'] + pearson_def(l,fl_bet_cen)

for key in mitja025008noi:
    mitja025008noi[key] = mitja025008noi[key]/25

mitja06008i = {'bet_cen':0,'com_cen':0, 'clo_cen':0, 'eig_cen':0, 'pg_rk':0, 'fl_clo_cen':0, 'fl_bet_cen':0}
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_networkx/Pin_=_0,6_Pout_=_0,08/comiguals/
        ↪ grafi06008'+str(i)+'.edgelist',nodetype = int, data = False)
```

```
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
    pg_rk = nx.pagerank(G)
    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    com = [[] for x in range(4)]
    for i in xrange(0,32):
        com[0].append(i)
        com[1].append(i+32)
        com[2].append(i+64)
        com[3].append(i+96)
    l = []
    for v in nx.nodes(G):
        l.append(outin(com,v,G))
    mitja06008i['bet_cen'] = mitja06008i['bet_cen'] + pearson_def(l,bet_cen)
    mitja06008i['com_cen'] = mitja06008i['com_cen'] + pearson_def(l,com_cen)
    mitja06008i['clo_cen'] = mitja06008i['clo_cen'] + pearson_def(l,clo_cen)
    mitja06008i['eig_cen'] = mitja06008i['eig_cen'] + pearson_def(l,eig_cen)
    mitja06008i['pg_rk'] = mitja06008i['pg_rk'] + pearson_def(l,pg_rk)
    mitja06008i['fl_clo_cen'] = mitja06008i['fl_clo_cen'] + pearson_def(l,fl_clo_cen)
    mitja06008i['fl_bet_cen'] = mitja06008i['fl_bet_cen'] + pearson_def(l,fl_bet_cen)

for key in mitja06008i:
    mitja06008i[key] = mitja06008i[key]/25

mitja06008noi = {'bet_cen':0,'com_cen':0, 'clo_cen':0, 'eig_cen':0, 'pg_rk':0, 'fl_clo_cen':0, 'fl_bet_cen':0}
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs de prova/Amb networkx/Pin = 0,6 Pout = 0,08/comnoiguals
        ↪ /grafnoi06008'+str(i)+'.edgelist',nodetype = int, data = False)
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
    pg_rk = nx.pagerank(G)
    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    com = [[] for x in range(4)]
    for i in xrange(0,16):
        com[0].append(i)
        com[1].append(i+16)
        com[2].append(i+32)
        com[2].append(i+48)
        com[3].append(i+64)
        com[3].append(i+80)
        com[3].append(i+96)
        com[3].append(i+112)
    l = []
    for v in nx.nodes(G):
        l.append(outin(com,v,G))
    mitja06008noi['bet_cen'] = mitja06008noi['bet_cen'] + pearson_def(l,bet_cen)
    mitja06008noi['com_cen'] = mitja06008noi['com_cen'] + pearson_def(l,com_cen)
    mitja06008noi['clo_cen'] = mitja06008noi['clo_cen'] + pearson_def(l,clo_cen)
    mitja06008noi['eig_cen'] = mitja06008noi['eig_cen'] + pearson_def(l,eig_cen)
    mitja06008noi['pg_rk'] = mitja06008noi['pg_rk'] + pearson_def(l,pg_rk)
    mitja06008noi['fl_clo_cen'] = mitja06008noi['fl_clo_cen'] + pearson_def(l,fl_clo_cen)
```

```
    mitja06008noi['fl_bet_cen'] = mitja06008noi['fl_bet_cen'] + pearson_def(l,fl_bet_cen)

for key in mitja06008noi:
    mitja06008noi[key] = mitja06008noi[key]/25

mitja025001i = {'bet_cen':0,'com_cen':0, 'clo_cen':0, 'eig_cen':0, 'pg_rk':0, 'fl_clo_cen':0, 'fl_bet_cen':0}
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs de prova/Amb networkx/Pin = 0,25 Pout = 0,01/comiguals/
        ↪ grafi025001'+str(i)+'.edgelist', nodetype = int, data = False)
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
    pg_rk = nx.pagerank(G)
    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    com = [[] for x in range(4)]
    for i in xrange(0,32):
        com[0].append(i)
        com[1].append(i+32)
        com[2].append(i+64)
        com[3].append(i+96)
    l = []
    for v in nx.nodes(G):
        l.append(outin(com,v,G))
    mitja025001i['bet_cen'] = mitja025001i['bet_cen'] + pearson_def(l,bet_cen)
    mitja025001i['com_cen'] = mitja025001i['com_cen'] + pearson_def(l,com_cen)
    mitja025001i['clo_cen'] = mitja025001i['clo_cen'] + pearson_def(l,clo_cen)
    mitja025001i['eig_cen'] = mitja025001i['eig_cen'] + pearson_def(l,eig_cen)
    mitja025001i['pg_rk'] = mitja025001i['pg_rk'] + pearson_def(l,pg_rk)
    mitja025001i['fl_clo_cen'] = mitja025001i['fl_clo_cen'] + pearson_def(l,fl_clo_cen)
    mitja025001i['fl_bet_cen'] = mitja025001i['fl_bet_cen'] + pearson_def(l,fl_bet_cen)

for key in mitja025001i:
    mitja025001i[key] = mitja025001i[key]/25

mitja025001noi = {'bet_cen':0,'com_cen':0, 'clo_cen':0, 'eig_cen':0, 'pg_rk':0, 'fl_clo_cen':0, 'fl_bet_cen':0}
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs de prova/Amb networkx/Pin = 0,25 Pout = 0,01/
        ↪ comnoiguals/grafnoi025001'+str(i)+'.edgelist',nodetype = int, data = False)
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
    pg_rk = nx.pagerank(G)
    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    com = [[] for x in range(4)]
    for i in xrange(0,16):
        com[0].append(i)
        com[1].append(i+16)
        com[2].append(i+32)
        com[2].append(i+48)
        com[3].append(i+64)
        com[3].append(i+80)
        com[3].append(i+96)
        com[3].append(i+112)
```

```
    l = []
    for v in nx.nodes(G):
        l.append(outin(com,v,G))
    mitja025001noi['bet_cen'] = mitja025001noi['bet_cen'] + pearson_def(l,bet_cen)
    mitja025001noi['com_cen'] = mitja025001noi['com_cen'] + pearson_def(l,com_cen)
    mitja025001noi['clo_cen'] = mitja025001noi['clo_cen'] + pearson_def(l,clo_cen)
    mitja025001noi['eig_cen'] = mitja025001noi['eig_cen'] + pearson_def(l,eig_cen)
    mitja025001noi['pg_rk'] = mitja025001noi['pg_rk'] + pearson_def(l,pg_rk)
    mitja025001noi['fl_clo_cen'] = mitja025001noi['fl_clo_cen'] + pearson_def(l,fl_clo_cen)
    mitja025001noi['fl_bet_cen'] = mitja025001noi['fl_bet_cen'] + pearson_def(l,fl_bet_cen)

for key in mitja025001noi:
    mitja025001noi[key] = mitja025001noi[key]/25

print "0,25/0,08 amb comunitats iguals:"
for key in mitja025008i:
    print key, ' ', mitja025008i[key], ' '
print ("\n")

print "0,25/0,08 amb comunitats diferents:"
for key in mitja025008noi:
    print key, ' ',mitja025008noi[key], ' '
print ("\n")

print "0,6/0,08 amb comunitats iguals:"
for key in mitja06008i:
    print key, ' ', mitja06008i[key], ' '
print ("\n")

print "0,6/0,08 amb comunitats diferents:"
for key in mitja06008noi:
    print key, ' ', mitja06008noi[key], ' '
print ("\n")

print "0,25/0,01 amb comunitats iguals:"
for key in mitja025001i:
    print key, ' ', mitja025001i[key], ' '
print ("\n")

print "0,25/0,01 amb comunitats diferents:"
for key in mitja025001noi:
    print key, ' ', mitja025001noi[key], ' '
print ("\n")
```

Grafs LFR

```
mitja02 = {'bet_cen':0,'com_cen':0, 'clo_cen':0, 'eig_cen':0, 'pg_rk':0, 'fl_clo_cen':0, 'fl_bet_cen':0}
rang = 0
for i in xrange(1,26):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_LFR_benchmark/mu_=_0,2/graf'+str(i)+'/
        ↪ network.dat',nodetype = int, data = False)
    s = open('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_LFR_benchmark/mu_=_0,2/graf'+str(i)+'/community.dat
        ↪ ', 'r')
    veccom = numpy.zeros((129,),dtype=numpy.int)
```

```
    maxi = 0
    for line in s:
        a = line.split()
        a = [int(x) for x in a]
        veccom[a[0]] = a[1]
        if a[1] > maxi:
            maxi = a[1]
    com = [[] * 1 for k in range(maxi)]
    for j in xrange(1,129):
        com[veccom[j]-1].append(j)
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
    pg_rk = nx.pagerank(G)
    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    l = []
    maxo = 0
    mini = 100
    for v in nx.nodes(G):
        a = outin(com,v,G)
        l.append(a)
        if (a > maxo):
            maxo = a
        if (a < mini):
            mini = a
    rang = rang + maxo-mini
    mitja02['bet_cen'] = mitja02['bet_cen'] + pearson_def(l,bet_cen)
    mitja02['com_cen'] = mitja02['com_cen'] + pearson_def(l,com_cen)
    mitja02['clo_cen'] = mitja02['clo_cen'] + pearson_def(l,clo_cen)
    mitja02['eig_cen'] = mitja02['eig_cen'] + pearson_def(l,eig_cen)
    mitja02['pg_rk'] = mitja02['pg_rk'] + pearson_def(l,pg_rk)
    mitja02['fl_clo_cen'] = mitja02['fl_clo_cen'] + pearson_def(l,fl_clo_cen)
    mitja02['fl_bet_cen'] = mitja02['fl_bet_cen'] + pearson_def(l,fl_bet_cen)

for key in mitja02:
    mitja02[key] = mitja02[key]/25
print "rang per mu = 0,2", ' ', rang/25
print ("\n")

mitja05 = {'bet_cen':0,'com_cen':0, 'clo_cen':0, 'eig_cen':0, 'pg_rk':0, 'fl_clo_cen':0, 'fl_bet_cen':0}
rang = 0
for i in xrange(1,26):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_LFR_benchmark/mu_=_0,5/graf'+str(i)+'/
        ↪ network.dat',nodetype = int, data = False)
    s = open('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_LFR_benchmark/mu_=_0,5/graf'+str(i)+'/community.dat
        ↪ ', 'r')
    veccom = numpy.zeros((129,),dtype=numpy.int)
    maxi = 0
    for line in s:
        a = line.split()
        a = [int(x) for x in a]
        veccom[a[0]] = a[1]
        if a[1] > maxi:
            maxi = a[1]
    com = [[] * 1 for k in range(maxi)]
```

```
    for j in xrange(1,129):
        com[veccom[j]-1].append(j)
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
    pg_rk = nx.pagerank(G)
    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    l = []
    maxo = 0
    mini = 100
    for v in nx.nodes(G):
        a = outin(com,v,G)
        l.append(a)
        if (a > maxo):
            maxo = a
        if (a < mini):
            mini = a
    rang = rang + maxo-mini
    mitja05['bet_cen'] = mitja05['bet_cen'] + pearson_def(l,bet_cen)
    mitja05['com_cen'] = mitja05['com_cen'] + pearson_def(l,com_cen)
    mitja05['clo_cen'] = mitja05['clo_cen'] + pearson_def(l,clo_cen)
    mitja05['eig_cen'] = mitja05['eig_cen'] + pearson_def(l,eig_cen)
    mitja05['pg_rk'] = mitja05['pg_rk'] + pearson_def(l,pg_rk)
    mitja05['fl_clo_cen'] = mitja05['fl_clo_cen'] + pearson_def(l,fl_clo_cen)
    mitja05['fl_bet_cen'] = mitja05['fl_bet_cen'] + pearson_def(l,fl_bet_cen)

print "rang per mu = 0,2", ' ', rang/25
print ("\n")


for key in mitja05:
    mitja05[key] = mitja05[key]/25

print "mu = 0,2"
for key in mitja02:
    print key,' ',mitja02[key],' '
print ("\n")

print "mu = 0,5"
for key in mitja05:
    print key,' ',mitja05[key],' '
print ("\n")
```

Grafs regulars amb comunitats

```
mitja001 = {'bet_cen':0,'com_cen':0, 'clo_cen':0, 'eig_cen':0, 'pg_rk':0, 'fl_clo_cen':0, 'fl_bet_cen':0}
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_comunitats_regulars/pout_=_0,0104/grafreg001
        ↪ '+str(i)+'.edgelist',nodetype = int, data = False)
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
```

```
    pg_rk = nx.pagerank(G)
    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    com = [[] for x in range(4)]
    for i in xrange(0,32):
        com[0].append(i)
        com[1].append(i+32)
        com[2].append(i+64)
        com[3].append(i+96)
    l = []
    for v in nx.nodes(G):
        l.append(outin(com,v,G))
    mitja001['bet_cen'] = mitja001['bet_cen'] + pearson_def(l,bet_cen)
    mitja001['com_cen'] = mitja001['com_cen'] + pearson_def(l,com_cen)
    mitja001['clo_cen'] = mitja001['clo_cen'] + pearson_def(l,clo_cen)
    mitja001['eig_cen'] = mitja001['eig_cen'] + pearson_def(l,eig_cen)
    mitja001['pg_rk'] = mitja001['pg_rk'] + pearson_def(l,pg_rk)
    mitja001['fl_clo_cen'] = mitja001['fl_clo_cen'] + pearson_def(l,fl_clo_cen)
    mitja001['fl_bet_cen'] = mitja001['fl_bet_cen'] + pearson_def(l,fl_bet_cen)

for key in mitja001:
    mitja001[key] = mitja001[key]/25

mitja01 = {'bet_cen':0,'com_cen':0, 'clo_cen':0, 'eig_cen':0, 'pg_rk':0, 'fl_clo_cen':0, 'fl_bet_cen':0}
for i in xrange(0,25):
    G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs_de_prova/Amb_comunitats_regulars/pout_=_0,1041/grafreg01'
        ↪ +str(i)+'.edgelist',nodetype = int, data = False)
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
    pg_rk = nx.pagerank(G)
    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    com = [[] for x in range(4)]
    for i in xrange(0,32):
        com[0].append(i)
        com[1].append(i+32)
        com[2].append(i+64)
        com[3].append(i+96)
    l = []
    for v in nx.nodes(G):
        l.append(outin(com,v,G))
    mitja01['bet_cen'] = mitja01['bet_cen'] + pearson_def(l,bet_cen)
    mitja01['com_cen'] = mitja01['com_cen'] + pearson_def(l,com_cen)
    mitja01['clo_cen'] = mitja01['clo_cen'] + pearson_def(l,clo_cen)
    mitja01['eig_cen'] = mitja01['eig_cen'] + pearson_def(l,eig_cen)
    mitja01['pg_rk'] = mitja01['pg_rk'] + pearson_def(l,pg_rk)
    mitja01['fl_clo_cen'] = mitja01['fl_clo_cen'] + pearson_def(l,fl_clo_cen)
    mitja01['fl_bet_cen'] = mitja01['fl_bet_cen'] + pearson_def(l,fl_bet_cen)

for key in mitja01:
    mitja01[key] = mitja01[key]/25

print "0,01"
for key in mitja001:
    print key, '_', mitja001[key], '_'
```

```
print ("\n")

print "0,1"
for key in mitja01:
    print key, ' ',mitja01[key], ' '
print ("\n")
```

Xarxes extretes de dades empíriques

```
def calcula(G):
    bet_cen = nx.betweenness_centrality(G, normalized=True)
    com_cen = nx.communicability_centrality(G)
    clo_cen = nx.closeness_centrality(G, normalized=True)
    eig_cen = nx.eigenvector_centrality_numpy(G)
    pg_rk = nx.pagerank(G)

    fl_clo_cen = nx.current_flow_closeness_centrality(G)
    fl_bet_cen = nx.current_flow_betweenness_centrality(G)
    com = louvain(G)

    l = []
    for v in nx.nodes(G):
        l.append(outin(com,v,G))
    bet_cenl = []
    com_cenl = []
    clo_cenl = []
    eig_cenl = []
    pg_rkl = []
    fl_clo_cenl = []
    fl_bet_cenl = []
    for key in nx.nodes(G):
        bet_cenl.append(bet_cen[key])
        com_cenl.append(com_cen[key])
        clo_cenl.append(clo_cen[key])
        eig_cenl.append(eig_cen[key])
        pg_rkl.append(pg_rk[key])
        fl_clo_cenl.append(fl_clo_cen[key])
        fl_bet_cenl.append(fl_bet_cen[key])

    res = {'bet_cen' : pearson_def(l,bet_cenl), 'com_cen' :pearson_def(l,com_cenl), 'clo_cen' : pearson_def(l,clo_cenl), '
        ↪ eig_cen' : pearson_def(l,eig_cenl), 'pg_rk':pearson_def(l,pg_rkl), 'fl_clo_cen':pearson_def(l,fl_clo_cenl), 'fl_bet_cen'
        ↪ : pearson_def(l,fl_bet_cenl)}

    return res

G = nx.read_gml('/home/aqm/Desktop/TFG/retfg/USAir-LesMis/USAir-LesMis/USAir2001.gml')
res = calcula(G)
print "Xarxa aeroports", ("\n")
for key in res:
    print key, ' ', res[key], ' '

G = nx.read_edgelist('/home/aqm/Desktop/TFG/retfg/prova-networkX/prova-networkX/asme-net.dat', nodetype = int)
res = calcula(G)
print "Xarxa asme", ("\n")
for key in res:
```

```
    print key, '␣', res[key], '␣'

G = nx.read_edgelist('/home/aqm/Desktop/TFG/retfg/prova-networkX/prova-networkX/na-net.dat', nodetype = int)
res = calcula(G)
print "Xarxa␣na", ("\n")
for key in res:
    print key, '␣', res[key], '␣'

G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs␣de␣prova/email/email.txt', nodetype=int, data = False)
res = calcula(G)
print "Xarxa␣email", ("\n")
for key in res:
    print key, '␣', res[key], '␣'

G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs␣de␣prova/jazz.net', nodetype=int, data = False)
res = calcula(G)
print "Xarxa␣jazz", ("\n")
for key in res:
    print key, '␣', res[key], '␣'


G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs␣de␣prova/dolphins/out.dolphins', data = False, nodetype = int)
res = calcula(G)
print "dolphins", ("\n")
for key in res:
    print key, '␣', res[key], '␣'


G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs␣de␣prova/moreno_beach/out.moreno_beach_beach', data = False,
    ↪ nodetype = int)
res = calcula(G)
print "windsurf", ("\n")
for key in res:
    print key, '␣', res[key], '␣'

G = nx.read_edgelist('/home/aqm/Desktop/TFG/Grafs␣de␣prova/moreno_train/out.moreno_train_train', data = False,
    ↪ nodetype = int)
res = calcula(G)
print "Terroristes␣11-S", ("\n")
for key in res:
    print key, '␣', res[key], '␣'
```