

FUNCIONALIDAD ANALÍTICA Y OPTIMIZACIÓN EN EL ANÁLISIS DE REDES

Comparativa entre Neo4j, networkx y igraph

November 6, 2019

Contents

1	Introducción	7
I	Simulación de redes	9
2	Métodos de simulación	9
2.1	En networkx (Clásicos)	9
2.1.1	Graph Atlas	9
2.1.2	Balanced Tree	9
2.1.3	Barbell Graph	10
2.1.4	Complete Graph	10
2.1.5	Complete multipartite graph	10
2.1.6	Circular Ladder Graph	10
2.1.7	Circulant Graph	10
2.1.8	Cycle Graph	11
2.1.9	Dorogovtsev, Goltsev, Mendes Graph	11
2.1.10	Empty Graph	12
2.1.11	Full Rary Tree	12
2.1.12	Ladder Graph	12
2.1.13	Lollipop Graph	12
2.1.14	Null Graph	13
2.1.15	Star Graph	13
2.1.16	Trivial Graph	13
2.1.17	Turan Graph	13
2.1.18	Wheel Graph	13

2.2	Otros de Networkx	14
2.2.1	Expanders	14
2.2.2	Lattice Graph	14
2.2.3	Small	14
2.2.4	Random Graphs	16
2.2.5	Divergencia duplicativa	17
2.2.6	Secuencia gradual	17
2.2.7	Random Clustered	18
2.2.8	Dirigidos	18
2.2.9	Geométricos	19
2.2.10	Grafos de línea	19
2.2.11	Grafos de Ego	20
2.2.12	Grafo Estocástico	20
2.2.13	Intersección	21
2.2.14	Redes Sociales	21
2.2.15	Comunidad	22
2.2.16	Espectral	22
2.2.17	Árboles	23
2.2.18	Triadas	23
2.2.19	Secuencia gradual conjunta	24
2.2.20	Mycielski	24
2.3	En igraph	24
2.3.1	Deterministas	25
2.3.2	Aleatorios	26
2.4	Modelos notables	27
2.4.1	Erdős-Rényi	27
2.4.2	De Bruijn	29
2.5	Neo4j	30
2.5.1	Neo4j SandBox	30
2.5.2	Integraciones de Neo4j	32

II Medidas de centralidad 36

3 Definición y computación de las medidas 36

3.1	En igraph	36
3.1.1	Burt's constraint [IG]	36
3.1.2	Grado máximo [IG]	37
3.1.3	Strength de un nodo [IG]	37
3.1.4	Kleinberg's Hub Scores [IG]	37
3.1.5	Kleinberg's Authority Scores [IG]	39
3.1.6	Métodos de estimación de centralidad [IG]	40
3.2	En networkx	40
3.2.1	VoteRank [NX]	40
3.2.2	Closeness (Current-Flow based) [NX]	40
3.2.3	Betweenness (current-flow based) [NX]	41

3.2.4	Communicability Betweenness [NX]	42
3.2.5	Load Centrality [NX]	43
3.2.6	Percolation (<i>filtración</i>) [NX]	44
3.2.7	Second order centrality [NX]	45
3.2.8	Reaching centrality [NX]	46
3.2.9	Subgraph centrality [NX]	47
3.2.10	Índice de Estrada [NX]	47
3.3	En Neo4j	48
3.3.1	Article Rank [N4J]	48
3.4	Compartidas	49
3.4.1	Pagerank [IG, N4J]	49
3.4.2	Degree [IG, NX, N4J]	50
3.4.3	Closeness Centrality [IG, NX, N4J]	50
3.4.4	Betweenness centrality [IG, NX, N4J]	50
3.4.5	Centralidad a través de los autovectores [IG, NX, N4J]	51
3.4.6	Harmonic centrality [NX, N4J]	51
3.5	Otras medidas de centralidad	52
3.5.1	ClusterRank	52
3.5.2	Rumor-source	52
3.6	Librería <i>CINNA</i> en R	52
4	Cómo escoger una medida de centralidad	54
III	Detección de comunidades	55
5	Descripción de los algoritmos	55
5.1	En igraph	55
5.1.1	Algoritmo basado en física estadística (statistical mechanics) [IG]	55
5.1.2	Algoritmo basado en los autovectores de las matrices [IG]	56
5.1.3	Método Infomap [IG]	57
5.1.4	Algoritmo de la modularidad óptima [IG]	59
5.1.5	Algoritmo walktrap [IG]	59
5.2	En networkx	61
5.2.1	Bipartición de Kernighan-Lin [NX]	61
5.2.2	K-clique comunidades a través de filtración [NX]	62
5.2.3	Fluid communities (asincronizado) [NX]	64
5.3	En Neo4j	65
5.3.1	Componentes conectados [N4J]	65
5.3.2	Componentes conectados fuertemente [N4J]	65
5.3.3	Conteo de triángulos / Coeficiente cluster [N4J]	66
5.3.4	Triadas equilibradas [N4J]	67
5.4	Algoritmos compartidos	67
5.4.1	Algoritmo multinivel o de Louvain [IG, N4J]	67

5.4.2	Algoritmo basado en la betweenness (intermediación) [IG, NX]	69
5.4.3	Método de la propagación de etiquetas [IG, NX, N4J]	70
5.4.4	Algoritmo fastgreegy [IG, NX]	71
6	Comparativa tiempo computacional	73
IV	Conectividad de una red	74
7	Homofilia (o asortatividad)	74
7.1	Definición	74
7.2	Medida	74
7.2.1	Coeficiente de asortatividad	74
7.2.2	Conectividad de vecino	75
7.2.3	Asortatividad Local	75
7.3	Computación	75
8	Otros	76
8.1	Multiplicidad	76
8.2	Mutualidad o reciprocidad	76
8.3	Network Closure	76
8.4	Propinquity	76
V	Otras propiedades	77
9	Descriptivas	77
9.1	Tamaño	77
9.2	Densidad	77
9.3	Grado medio	77
9.4	Camino más corto característico	78
10	De distribución	78
10.1	Puentes	78
10.1.1	Bridge-finding a través del algoritmo de Tarjan	78
10.1.2	Bridge-finding con descomposición en cadena	79
10.2	Structural Holes	80
10.2.1	Definición	80
10.2.2	Medida a través del tamaño efectivo	80
10.2.3	Medida a través de la restricción	81
10.3	Fuerza del vínculo o <i>Tie Strength</i>	81
10.4	Eficiencia	81
10.4.1	Definiciones	81
10.4.2	Aplicaciones	82
10.4.3	Otras nociones de eficiencia en redes	82

11 Otros algoritmos en networkx	83
 VI Problemas en el análisis de redes	 85
12 Enunciado y <i>solución</i> de los problemas	85
12.1 De enumeración	85
12.2 Subgrafos, subgrafos inducidos y menores	85
12.3 Coloreado de grafos	86
12.4 Problemas de ruta	87
12.4.1 Siete puentes de Königsberg	87
12.4.2 Problema del camino hamiltoniano	87
12.4.3 Mínimo spanning tree	88
12.4.4 Problema de inspección de la ruta	89
12.4.5 Camino más corto	90
12.4.6 Problema del árbol de Steiner	91
12.4.7 Problema del vendedor	91
12.5 Problemas de visibilidad	95
12.6 Problemas de descomposición	95
12.6.1 Factorización de un grafo	96
12.6.2 Arboricidad	96
12.7 Flow Network	97
12.7.1 Flujo máximo	97
12.7.2 Otros	99
12.8 Análisis del Camino Crítico y Evaluación de Programa (PERT) .	99
12.8.1 Análisis del camino crítico	99
12.8.2 Evaluación de programa (PERT)	99
12.9 Otros problemas	100
12.9.1 Problema de localización	100
12.9.2 Problema de transporte	100
 13 Propagación de contenido	 101
13.1 El modelo SIR	101
13.2 La ecuación maestra	102
 VII Análisis dinámico de redes	 104
14 Introducción al análisis dinámico	104
14.1 Sistema dinámico de grafos	104
 15 Predicción de links	 105
15.1 Introducción	105
15.2 Definición del problema	106
15.3 Medidas de similitud	106
15.3.1 Distancia en el grafo	106

15.3.2	Vecinos comunes	107
15.3.3	Coeficiente de Jaccard	107
15.3.4	Adamic-Adar	107
15.3.5	Conexión preferencial	108
15.3.6	Conteo del camino alisado exponencialmente de Katz	108
15.3.7	Hitting time	108
15.3.8	Rooted Pagerank	109
15.3.9	Friends-measure	109
15.3.10	Resource Allocation	110
15.3.11	Common Neighbors	110
15.3.12	Inter-cluster common neighbors	110
15.3.13	Total neighbors	111
15.3.14	Otras medidas de similitud	111
15.4	Computación de los algoritmos	112
VIII	Introducción a redes cuánticas	113
16	Redes cuánticas sociales	113
16.1	Introducción	113
16.2	Redes cuánticas sociales	114
16.3	Ejemplo de mejora	116
IX	Anti money-laundering	117
17	Teoría del AML	117
18	Enfoque de redes del AML	121
19	Propuestas de integración	123
19.1	Sistemas de la literatura	123
19.2	¿Qué sistemas se podrían implantar teniendo en cuenta el enfoque de redes?	126

1 Introducción

El objetivo principal de este documento será describir y explorar la capacidad de análisis que tres herramientas distintas tienen sobre el análisis de redes en base a las funcionalidades que tienen integradas.

Estas tres herramientas son dos librerías (para Python, aunque disponibles en más lenguajes), `networkx` y `igraph`; y un software de base de datos orientada a grafos, Neo4j, aunque con cierta capacidad analítica.

En una primera parte, se tratará brevemente los métodos de simulación de redes en las herramientas mencionadas. En los problemas que nos ocupan los grafos a analizar generalmente vienen dados por lo que, aunque en un contexto de desarrollo/aprendizaje esta funcionalidad es muy útil para, por ejemplo, probar nuevos métodos de análisis u optimización, en los casos reales su uso será realmente marginal.

Por ello, el objetivo central del trabajo será la descripción de métodos para el análisis, desarrollados en las partes II, III, IV y V, que ocupan físicamente el mayor espacio del documento: propiedades descriptivas tales como detección de comunidades o medición de la centralidad.

En la siguiente parte, la número VI, se dan problemas (y soluciones, tanto analíticas como algorítmicas) recurrentes en optimización de redes, un campo del análisis de redes que en un principio no iba a ser central en el documento, pero que podría llegar a aportar elementos relevantes para un conocimiento global de la materia si se decidiese profundizar en este área o, mismamente, para una mejor aprehensión del análisis como tal.

Finalmente, con el fin de dar cuenta de la existencia de caminos explorables, se añaden unas breves notas acerca de dos subcampos lo suficientemente emergentes en el análisis de redes, como es el análisis dinámico de redes, ya asentado en la disciplina y las redes cuánticas.

Como puede comprobarse, el objetivo del trabajo siempre trató de ser una aproximación a los medios que componen el estado del arte en el análisis de redes, es decir, a *lo que puede llegar a hacerse* a la hora de enfrentarnos a un problema, más que a la explicación detallada de *cómo se hace lo que usualmente se hace*.

En este sentido, priman las explicaciones teóricas sobre las técnicas y es por ello también que hay una gran cantidad de listas durante el documento, comenzando por un índice de seis páginas y una bibliografía de otras tantas. El objetivo de esto no es otro que permitir la posibilidad de utilización del trabajo

como guía para la *inspiración* ante un problema y, a partir de ello, si se necesita, profundizar a través de las referencias o de internet directamente en el método cuestionado o en sus apartados más técnicos. Por otra parte, el desarrollo más o menos detallado de esas listas *interiores* no merece la pena por dos razones: el fácil acceso y comprensión de la información en la red y cierta imposibilidad *física*, en tiempo y en espacio, que multiplicaría la extensión del documento, dejando las partes centrales como menores en extensión.

En la descripción de la computación se desarrolla más la parte de networkx que la de igraph por dos razones: tiene *alguna* funcionalidad más y, sobre todo, su documentación está mucho mejor preparada, limpia y ordenada¹.

Para terminar la introducción, dos apuntes más irrelevantes. En general, grafo y red se utilizan como sinónimos. Por otra parte, en la descripción de las funciones se omiten los parámetros menos relevantes, con el objetivo de concreción descrito en párrafos anteriores.

¹En igraph para R y, sobre todo, para C la información está mejor estructurada, pero no es exactamente igual la librería para todos los lenguajes.

Part I

Simulación de redes

En esta primera sección se tratará de comparar el alcance de las herramientas disponibles en cada uno de los softwares estudiados. En la literatura podemos encontrar modelos teóricos de redes, en los que se estudian formas y propiedades inherentes a ciertas estructuras topológicas.

Estos modelos sirven como base para el estudio y entendimiento de las interacciones a través de redes complejas del mundo real. A través de la generación *aleatoria* pero dirigida de estos grafos pueden estudiarse propiedades en base a la comparación con las redes de la realidad.

Seguidamente se enlistarán todas las funciones de simulación que hay en los distintas opciones de software, pero debido a la gran cantidad de opciones disponibles, sólo se describirán con cierto detalle los referentes a los grafos clásicos en networkx. Sin perjuicio de lo anterior, se darán breves apuntes sobre ciertas funcionalidades recogidas.

2 Métodos de simulación

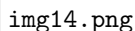
2.1 En networkx (Clásicos)

2.1.1 Graph Atlas

En esta función de networkx `graph_atlas(i)`, se introduce un valor entero i y se devuelve el grafo correspondiente a dicho entero en el libro *An Atlas of Graphs*, cuyo índice puede verse, o bien con la función `graph_atlas_g()` o en la referencia [65], entre los que se encuentran: grafos cúbicos conectados, grafos eulerianos, etc.

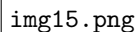
2.1.2 Balanced Tree

La función `balanced_tree(r,h)`, con r número de hijos que tiene cada nodo y h la altura de cada nodo, devuelve el árbol de dichas catacterísticas: h generaciones con r hijos de cada nodo.



2.1.3 Barbell Graph

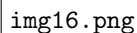
Esta función *barbell_graph(m1,m2)* devuelve dos grafos completos conectados por un camino. Dos grafos completos² K_{m_1} conectados por un camino P_{m_2} .

The diagram shows a barbell graph, which consists of two complete graphs K_{m_1} connected by a path P_{m_2} .

img15.png

2.1.4 Complete Graph

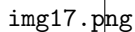
Crea un grafo completo K_n mediante *complete_graph(n)* con n el número de nodos.

The diagram shows a complete graph K_n , where every node is connected to every other node.

img16.png

2.1.5 Complete multipartite graph

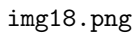
La función *graph_multipartite_graph(subset_sizes)* crea un grafo multipartito³ con los tamaños de cada subconjunto especificados en la tupla de *subset_sizes*.

The diagram shows a complete multipartite graph, where nodes are partitioned into subsets and every node in one subset is connected to every node in another subset.

img17.png

2.1.6 Circular Ladder Graph

Mediante la función *circular_ladder_graph(n)* CL_n de longitud n , consistente en dos n -ciclos cuyos n pares de nodos concéntricos están unidos por una arista.

The diagram shows a circular ladder graph CL_n , which consists of two concentric cycles of length n , with corresponding nodes connected by edges.

img18.png

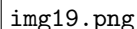
2.1.7 Circulant Graph

La función de networks *circulant_graph(n, offsets)* devuelve el grafo circulante $Ci_n(x_1, \dots, x_m)$ consistente en n nodos tales que el vértice con la etiqueta i está

²Un grafo es completo si todos sus nodos están conectados entre sí.

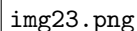
³Un grafo multipartito es aquel cuyos vértices pueden ser particionados en k conjuntos independientes.

conectado a aquellos con la etiqueta $i+x$ y $i-x \forall x \in (x_1, \dots, x_m)$, que el conjunto *offsets*.



2.1.8 Cycle Graph

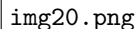
A través de `cycle_graph(n)` podemos crear el grafo cíclico C_n de n nodos conectados.



2.1.9 Dorogovtsev, Goltsev, Mendes Graph

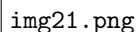
Esta función, a la que se le otorga como input el número de nodos n , crea el grafo jerárquicamente construido de Dorogovtsev, Goltsev y Mendes; que ellos llaman *Pseudofractal Scale-free Web*.

Básicamente, utilizan para modelizar redes aleatorias de tipo scale-free a través de un grafo determinista que tiene una distribución de los grados de cada nodo que sigue una ley potencial $\gamma = 1 + \frac{\ln 3}{\ln 2}$.



En la imagen se aprecia el esquema del crecimiento del grafo pseudofractal scale-free. El crecimiento comienza con una sola arista conectando dos vértices en $t = -1$. A cada paso, cada arista genera un nodo adicional, que es conectado a ambos nodos de la arista.

Además, demuestran que la matriz de adyacencia se comporta como sigue:



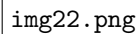
Esta imagen, en concreto, representa la estructura de la matriz de adyacencia para el grafo con $t = 2$ y $N_t = 15$. Las regiones en negro son los elementos unidad de la matriz, mientras en blanco son todo ceros. En las regiones en las demás regiones de la matriz simétrica, cada columna por encima de la diagonal principal solo contiene dos elementos no nulos.

2.1.10 Empty Graph

La función *empty_graph(n)* devuelve el grafo vacío de n nodos sin relaciones entre ellos.

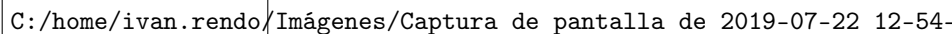
2.1.11 Full Rary Tree

Un árbol full-r-ary se puede definir como aquel donde todos los nodos que no sean hojas tienen exactamente r hijos y todos los niveles están *llenos* excepto para algunas posiciones del nivel inferior de tal manera que, si una hoja es faltante, todas a su derecha lo serán. La función *full_rary_tree(r,n)* indica r , factor de ramaje y n nodos.



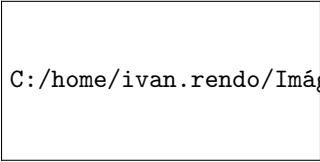
2.1.12 Ladder Graph

La función *ladder_graph(n)* devuelve el grafo *ladder* de longitud n , L_n . Esto es, dos caminos de n nodos con cada par conectado por una única arista.



2.1.13 Lollipop Graph

El siguiente tipo de grafo viene generado por la función *lollipop_graph(m,n)*. Un grafo lollipop es un grafo completo K_m al que se le conecta un camino P_n .



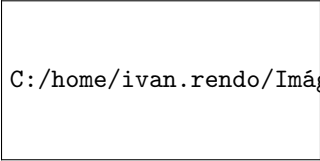
C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-22 12-56-50

2.1.14 Null Graph

Se devuelve directamente un grafo sin nodos ni aristas a través de *null_graph()*.

2.1.15 Star Graph

Un grafo estrella es aquel que tiene un nodo central y n nodos exteriores, a los que está conectado mediante una arista. La función es *star_graph(n)*.



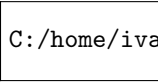
C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-22 13-09-19

2.1.16 Trivial Graph

La función *trivial_graph()* devuelve el grafo trivial que sólo posee un nodo y ninguna arista.

2.1.17 Turan Graph

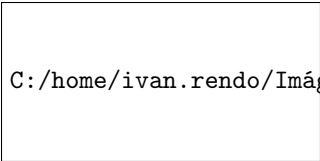
La función *turan_graph(n,r)* devuelve un grafo de Turan, es decir, un grafo completo multipartido con n nodos y r subconjuntos disjuntos.



C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-22 13-13-03

2.1.18 Wheel Graph

Un grafo rueda es aquel que contiene un ciclo de orden $n - 1$ y que, cada nodo en el ciclo se conecta a otro nodo del grafo, llamado *hub*.



C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-22 13-13-03

2.2 Otros de Networkx

2.2.1 Expanders

Se dice que un grafo es *expander* cuando es un grafo disperso con propiedades de alta conectividad, cuantificados utilizando vértices, aristas o expansión espectral. En networkx pueden crearse los siguientes:

- `margulis_gabber_galil_graph(n)`
- `chordal_cycle_graph(p)`

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-22 1

2.2.2 Lattice Graph

Un grafo tipo *lattice* o enrejado es un grafo cuyo dibujo, en algún espacio de \mathbb{R}^n en el que esté definido, forma una malla regular. En networkx existen:

- `grid_2d_graph(m,n,periodic)`
- `grid_graph(dim)`
- `hexagonal_lattice_graph(m,n,periodic)`
- `hypercube_graph(n)`
- `triangular_lattice_graph(m,n)`

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-22 13

2.2.3 Small

En esta amplia sección se encuentran grafos *famosos* pequeños y otros generadores compactos.

- `make_small_graph(graph_description)`
- `LCF_graph(n, shift_list, repeats)`
- `bull_graph()`
- `chvatal_graph()`

- *cubical_graph()*
- *desargues_graph()*
- *diamond_graph()*
- *dodecahedral_graph()*
- *frucht_graph()*
- *heawood_graph()*
- *hoffman_singleton_graph()*
- *house_graph()*
- *house_x_graph()*
- *icosahedral_graph()*
- *krackhardt_kite_graph()*
- *moebius_kantor_graph()*
- *octahedral_graph()*
- *pappus_graph()*
- *petersen_graph()*
- *sedgewick_maze_graph()*
- *tetrahedral_graph()*
- *truncated_cube_graph()*
- *truncated_tetrahedron_graph()*
- *tutte_graph()*

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-22 13-34-

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-22 13-34-

Grafo de Pappus (Papo de Alejandría) y *Krackhardt kite*, respectivamente.

2.2.4 Random Graphs

En esta sección se enlistan los generadores de grafos aleatorios que dependen de ciertos parámetros: en general, nodos y aristas y de una semilla. Además, pueden escogerse dirigidos o no dirigidos en *casi todos* los casos.

- `fast_gnp_random_graph(n,p)`
- `gnp_random_graph(n,p)`
- `dense_gnm_random_graph(n,m)`
- `gnm_random_graph(n,m)`
- `erdos_renyi_graph(n,p)`⁴

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-22 13-46-

Generado mediante Barabasi-Albert

- `binomial_graph(n,p)`
- `newman_watts_strogatz_graph(n,k,p)`
- `watts_strogatz_graph(n,k,p)`
- `connected_watts_strogatz_graph(n,k,p)`
- `random_regular_graph(d,n)`
- `barabasi_albert_graph(n,m)`
- `dual_barabasi_albert_graph(n,m1,m2,p)`
- `extended_barabasi_albert_graph(n,m,p,q)`
- `powerlaw_cluster_graph(n,m,p)`
- `random_kernel_graph(n,kernel_integral)`
- `random_lobster(n,p1,p2)`

⁴Fundamental en teoría de grafos. Por ello explicado en la sección 2.4, perteneciente a este capítulo.

- *random_shell_graph(constructor)*
- *random_powerlaw_tree(n,gamma)*
- *random_powerlaw_tree_sequence(n,gamma)*
- *random_kernel_graph(n,kernel_integral)*

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-22 13-46-

Generado mediante lobster

2.2.5 Divergencia duplicativa

Estas funciones, basadas en las redes de la biología, generan grafos basadas en el método de la *duplicación*: comienzan con un grafo inicial pequeño y duplican nodos y (parcialmente) aristas. Hay dos funciones para crearlas en networkx:

- *duplication_divergence_graph(n,p)*
- *partial_duplication_graph(N,n,p,q)*

2.2.6 Secuencia gradual

Las funciones de este tipo generan grafos de acuerdo a una secuencia de grados o una secuencia esperada de grados. Estas son las funciones que lo realizan:

- *configuration_model(deg_seq)*
- *directed_configuration_model(deg_seq)*
- *expected_degree_graph(w)*
- *havel_hakimi_graph(deg_seq)*
- *directed_havel_hakimi_graph(in_deg_seq)*
- *degree_sequence_tree(deg_seq)*
- *random_degree_sequence_graph(seq)*

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 08-57-

Grafos con secuencia gradual $\{3, 3, 3, 3, 3, 3\}$

2.2.7 Random Clustered

En este apartado se encuentra sólomente una función, que genera grafos dados un cierto grado y una secuencia de triángulos. El modelo genera un grafo aleatorio donde se incluyen aristas paralelas y conexiones con uno mismo para que las aristas -aleatoriamente- cumplan la secuencia dada.

Dicha secuencia es una lista de tuplas de enteros de forma: $[(d_{1i}, d_{1t}), \dots, (d_{ni}, d_{nt})]$, tal que un nodo u es miembro de d_{ut} triángulos (*triangle degree*) y tiene d_{ui} otras aristas (*independent edge degree*).

- `random_clustered_graph(joint_degree_sequence)`

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 09-13-

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 09-11-

Grafos generados con la lista $[(1, 0), (1, 1)(0, 1), (0, 1), (0, 1), (0, 2)]$

2.2.8 Dirigidos

Las funciones de este apartado sirven para crear grafos dirigidos, incluyendo redes crecientes (*growing networks*), y scale-free (*libre de escala*).

- `gn_graph(n)`
- `gnr_graph(n, p)` - p : probabilidad de redirección - *growing network*
- `gnc_graph(n)` - *growing network con copia*

- *random_k_out_graph(n,k,alpha)* - *k*: outdegree de cada nodo, *alpha*: peso inicial de cada nodo para ver
- *scalre_free_graph(n)*

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 09-29-

Grafo generado mediante gnc_graph(n=24)

2.2.9 Geométricos

Este grupo de funciones generan grafos con ciertas propiedades geométricas, donde las aristas se distribuyen uniformemente o según otra ley *geométrica*.

- *random_geometric_graph(n,radius)*
- *soft_random_geometric_graph(n,radius)*
- *geographical_threshold_graph(n,theta)*
- *waxman_graph(n,beta,alpha)*
- *navigable_small_world_graph(n)*
- *thresholded_random_geometric_graph(n)*

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 09-35-

Grafo generado por random_geometric_graph(15,0.7)

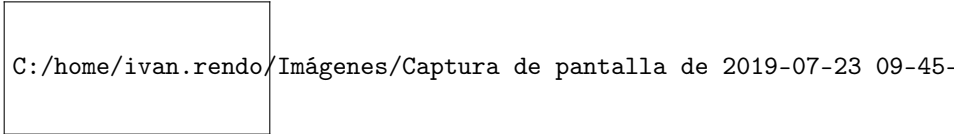
2.2.10 Grafos de línea

Funciones para generar grafos lineales⁵ (a partir de otros grafos).

⁵Un grafo de línea $L(G)$ de un grafo G no dirigido es un grafo que representa las adyacencias entre las aristas de G . Es decir, cumple dos condiciones:

1. cada nodo de $L(G)$ representa una arista de G , y
2. dos vértices de $L(G)$ son adyacentes si y sólo si sus aristas correspondientes comparten un punto final en común, i.e., son adyacentes en G .

- $line_graph(G)$
- $inverse_line_graph(G)$

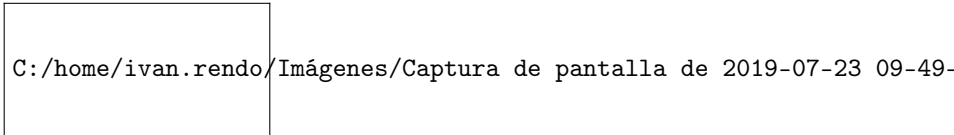


Grafo de línea generado a partir del grafo geométrico anterior.

2.2.11 Grafos de Ego

Un grafo de Ego también parte de un grafo dado G y de un nodo a escoger n . Se denomina grafo de Ego al subgrafo inducido de los vecinos de n centrados en él, dado cierto radio, predefinido en 1.

- $ego_graph(G, n, radius=1)$

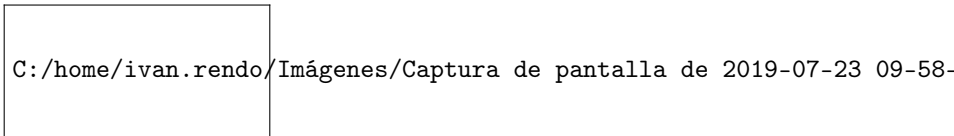


Grafo de ego centrado en $n=1$ y G geométrico anterior.

2.2.12 Grafo Estocástico

La función crea la representación *right*-estocástica⁶ de un grafo G dado (dirigido).

- $stochastic_graph(G)$



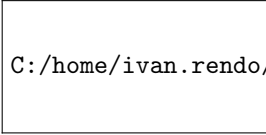
Representación estocástica de un grafo aleatorio dirigido con peso de arista unitario.

⁶Un grafo *right-estocástico* es un digrafo con pesos donde, para cada nodo, la suma de los pesos de todas las aristas salientes es 1. Es un concepto creado a partir de la matriz *right-estocástica* utilizada en cadenas de Markov.

2.2.13 Intersección

Este conjunto de funciones generan grafos de intersección⁷ de tres maneras: uniforme, aleatorio dado un tamaño k y puramente aleatorio.

- *uniform_random_intersection_graph(n, m, p)* - n : nodos del primer set, m : nodos del segundo set, p : lista con probabilidad de conectar nodos de cada atributo.
- *k_random_intersection_graph(n, m, k)*
- *general_random_intersection_graph(n, m, p)*



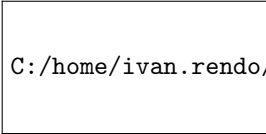
C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 10-11-11.png

Grafo generado por uniform_random_intersection_graph(6,15,0.25)

2.2.14 Redes Sociales

Aquí pueden encontrarse cuatro ejemplos clásicos (*en el sentido de famosos*) precargados de la literatura del análisis de redes sociales con los que poder trabajar.

- *karate_club_graph()*, creado por Zachary [69].
- *davis_southern_women_graph()*, de Davis y Gardner [70].
- *florentine_families_graph()*, de R. Breiger y P. Pattison [71].
- *les_miserables_graph()*, de Knuth, donde se muestra la relación entre los personajes de la obra *Los Miserables*. [72]



C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 10-24-11.png

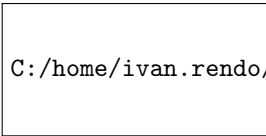
Grafo de R. Breiger y P. Pattison: Florentine Families.

⁷Un grafo de intersección es un grafo no dirigido formado por una familia de conjuntos S_i para $i = 0, 1, 2, \dots$ creando un nodo v_i para cada conjunto S_i y conectando dos nodos v_i y v_j por una arista siempre que los conjuntos dados cumplan $S_i \cap S_j \neq \emptyset$.

2.2.15 Comunidad

Con las siguientes funciones se generan grafos con ciertas características que son recurrentemente estudiados en el análisis de redes sociales.

- `caveman_graph(l,k)`, con l cliques de tamaño k
- `connected_caveman_graph(l,k)`
- `relaxed_caveman_graph(l,k,p)`, p : probabilidad de relación
- `random_partition_graph(sizes,p_in,p_out)`
- `planted_partition_graph(l,k,p_in,p_out)`, con l -particiones, p_{in} : probabilidad de relación intragrupo, p_{out} : probabilidad de relación intergrupo.
- `gaussian_random_partition_graph(n,s,v)`, s : tamaño medio del cluster, v : parámetro de forma.
- `ring_of_cliques(num_cliques,cliques_size)`
- `stochastic_block_model(sizes,p)`
- `windmill_graph(n,k)`



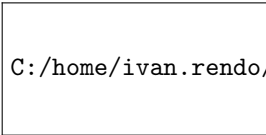
C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 10-32-10

Grafo generado por `caveman_graph(2,5)`

2.2.16 Espectral

Se trata de una función que realiza el algoritmo SGF, *Spectral Graph Forge*, que computa los autovectores de la matriz de adyacencia de un grafo G dado, filtra un porcentaje dado mediante un parámetro $\alpha \in (0, 1)$ y construye un grafo aleatorio con una estructura espectral similar.

- `spectral_graph_forge(G,alpha)`



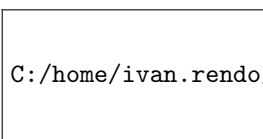
C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 10-38-10

`Spectral_graph_forge(G,0.5)` con G el caveman anterior.

2.2.17 Árboles

Estas funciones generan árboles, tanto de manera uniformemente aleatoria como para crear un Trie⁸ a partir de una lista de caminos.

- *random_tree(n)*
- *prefix_tree(paths)*



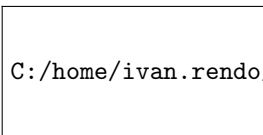
Grafo generado por random_tree(n=20)

2.2.18 Triadas

La función genera un grafo de triadas, esto es, los posibles digrafos en tres nodos.

La función es la siguiente:

- *triad_graph(triad_name⁹)*



Ejemplo de triada, en este caso '120U'

⁸Un trie es una estructura de datos de tipo árbol que permite la recuperación de información.

Formalmente se trata de un autómata finito determinista (S, Σ, T, s, A) llamado autómata finito determinista acíclico AFDA, que almacena un conjunto de cadenas E en el que:

- Σ es el alfabeto sobre el que se definen las cadenas.
- S es el conjunto de estados, cada uno representa un prefijo de E .
- La función de transición: $T : S \times \Sigma \rightarrow S$ definida tal que $T(x, \sigma) = x\sigma$ siempre que $\{x, x\sigma\} \in E$ e indefinida en otro caso.
- El estado inicial s corresponde a la cadena vacía λ .
- El conjunto de estados de aceptación $A \subseteq S$ es igual a E .

⁹Todas las opciones posibles para una triada son las siguientes: ('003', '012', '102', '021D', '021U', '021C', '111D', '111U', '030T', '030C', '201', '120D', '120U', '120C', '210', '300')

2.2.19 Secuencia gradual conjunta

Se encuentran dos funciones en esta sección, pero la primera sólo devuelve si una secuencia de grados es válida para generar un grafo (con la segunda función). Esta última genera un grafo aleatorio simple con el diccionario de grados dado.

- `is_valid_joint_degree(joint_degrees)`
- `joint_degree_graph(joint_degrees)`

2.2.20 Mycielski

Estas últimas funciones de networkx están relacionadas con los grafos de Mycielski¹⁰.

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 11-09-

- `mycielskian(G)` - devuelve el grafo de Mycielski de un grafo simple no dirigido G .
- `mycielskian_graph(n)` - genera el grafo de n nodos de Mycielski.

Nota:

Todas las funciones pueden incorporar una semilla y un formato de salida.

2.3 En igraph

En igraph existen métodos de generación no disponibles en networkx y viceversa pero, en general, los generadores son compartidos entre las dos librerías, sobre todo los más utilizados. Podemos dividir en dos tipos los generadores de grafos de igraph, deterministas y aleatorios. Se enlistarán a continuación las diferentes funciones.

¹⁰Un grafo de Mycielski es un grafo formado por construcción que preserva la propiedad de ser *triangle-free* pero incrementa en número cromático.

La construcción es la siguiente: dados n nodos de un grafo dado G , v_1, \dots, v_n , el grafo de Mycielski $\mu(G)$ añade $n+1$ nodos adicionales: un nodo u_i correspondiente a cada nodo v_i y un nodo extra w . Cada nodo u_i se relaciona por una arista con w , así que estos -conjuntamente- forman una estrella K_{1n} . Además, para cada arista $\overrightarrow{v_i v_j}$ en G , el grafo de Mycielski incluye dos aristas: $\overrightarrow{u_i v_j}$ y $\overrightarrow{v_i u_j}$.

2.3.1 Deterministas

- *Adjacency(matrix)* - genera un grafo a partir de su matriz de adyacencia.
- *Atlas(idx)* - genera el grado *idx* del Graph Atlas.
- *De Bruijn(n,m)* - genera un grafo de De Bruijn con parámetros *n* y *m*.¹¹
- *Full_Citation(n)* - Genera un grafo lleno con citación.
- *Famous(name)* - Reproduce un grafo famoso precargado de una lista:
 - Bull
 - Chvatal
 - Coxeter
 - Cubical
 - Diamond
 - Dodecahedral
 - Folkman
 - Franlin
 - Frucht
 - Grotzsch
 - Heawood
 - Herschel
 - House
 - HouseX
 - Icosahedral
 - Krackhardt_Kite
 - Levi
 - McGee
 - Meredith
 - Noperfectmatching
 - Nonline

¹¹Fundamental en teoría de grafos. Por ello explicado en la sección 2.4, perteneciente a este capítulo.

- Octahedral
 - Petersen
 - robertson
 - Smallestcyclingroup
 - Tetrahedral
 - Thomassen
 - Tutte
 - Uniquely3colorable
 - Walther
 - Zachary
- *Full(n)* - Genera un grafo lleno con n nodos.
 - *Isoclass(n, class)* - Genera un grafo para una clase isomorfa dada.
 - *Kautz(m,n)* - Genera un grafo de Kautz, donde la restricción es que cada dos etiquetas consecutivas, la cadena debe ser diferente.
 - *LCF(m,shifts)* - Genera un grafo Hamiltoniano 3-regular.
 - *Lattice(dim)* - Genera un grafo tipo reja regular.
 - *Ring(n)* - Genera un grafo tipo anillo de n nodos.
 - *Tree(n, children)* - Genera un árbol donde *casi* todos los nodos tienen el mismo número de hijos, *children*.
 - *Weighted_Adjacency(matrix)* - Genera un grafo a partir de su matriz de adyacencia.

2.3.2 Aleatorios

- *Asymmetric_Preference(n, type_dist_matrix, pref_matrix)* - genera un grafo basado en tipos de nodos asimétricos y probabilidades de conexión.
- *Barabasi(n,m)* - genera un grafo basado en el modelo Barabasi-Albert.
- *Degree_sequence(out, in=None)* - genera un grafo con las secuencias {in, out} de grados fijadas para cada nodo.
- *Erdos_Renyi(n,p,m)* - genera un grafo basado en el modelo Erdos-Renyi

- *Establishment*($n, k, type_dist, pref_matrix$) - Genera un grafo con un modelo creciente simple y una matriz de preferencia.
- *Forest_fire*(n, fw_prob) - genera un grafo basado en el modelo del incendio, un modelo *growing*.
- *Growing_Random*(n, m)
- *K_Regular*(n, k) - genera un grafo donde cada nodo tiene grado k .
- *Preference*($n, type_dist, pref_matrix$) - variante sin crecimiento del establishment.
- *Recent_Degree*($n, m, window$) - genera un grafo basado en un modelo estocástico donde la probabilidad de que una arista *gane* un nuevo nodo es proporcional a las aristas *ganadas* en cierta ventana temporal.
- *SBM*($n, pref_matrix, block_sizes$) - Genera un grafo basado en el modelo estocástico por bloques.
- *Static_Fitness*($m, fitness_out, fitness_in=None$) - Genera un grafo no-creciente con probabilidades de aristas proporcional al fitness de los nodos.
- *Static_Power_Law*($n, m, exponent_out, exponent_in=-1$) - Genera un grafo no-creciente que sigue una distribución de ley potencial prefijada.
- *Watts_Strogatz*($dim, size, nei, p$)

2.4 Modelos notables

2.4.1 Erdős-Rényi

El modelo Erdős-Rényi o *ER* es un método de generación aleatorio de grafos basado en la idea de que un nuevo nodo se enlaza con igual probabilidad al resto de la red. Se suele tomar como punto de partida en el estudio de las redes complejas o la simulación, pero no suele aplicarse debido a varias limitaciones; fundamentalmente que las redes reales que se comportan así son muy limitadas.

Si consideramos N nodos de una red sin conexiones y se conectan dos de ellos en cada una de las iteraciones de un proceso de M etapas, podemos intuir que, cuanto mayor sea M respecto de N mayor será la densidad y conectividad de la red. Además, en ese caso, la distribución de grado obtiene propiedades

específicas que se enlazan con la distribución de Poisson, por las características del proceso.

Nuestra intención es calcular la distribución de grado, para facilitar la comprensión de la estructura de la red. Comenzamos calculando el número total de parejas de N nodos en una red (N_p), que no es otro que el combinatorio.

$$N_p = \binom{N}{2}$$

El número de parejas enlazadas por el modelo es M , como hemos dicho, por lo que la probabilidad de que una pareja esté enlazada p_c es:

$$p_c = \frac{M}{N_p}$$

Si ahora tomamos un nodo al azar de la red, v_j , el número de nodos enlazados a pares que contuvieran a v_j sería $N - 1$, ya que dicho nodo se puede enlazar con los restantes nodos de la red; pero puede que en los M generados no estuviera v_j . Suponemos que está en k de ellas, que será su grado, y no está en $N - 1 - k$. La probabilidad de esto es:

$$P(k) = \binom{N-1}{k} (p_c)^k (1-p_c)^{N-1-k}$$

Que es una distribución binomial. Si consideramos N y M suficientemente grande como para asumir $N, M \rightarrow \infty$, $P(k)$ se convierte en una distribución de Poisson de tasa z :

$$P(k) = e^{-z} \frac{z^k}{k!}$$

Con valor $z = \frac{2M}{N} \in \mathbb{R}$, proveniente de $p_c = \frac{2M}{N(N-1)} = \frac{z}{N-1}$.

El problema de su inaplicabilidad proviene de esto: la distribución de grado de la realidad tiende a ser exponencial $P(k) = Ce^{-\alpha k}$ y no de Poisson. La aplicabilidad es un problema también corregido por las redes de libre escala de Barabasi, que alcanzan $P(k) = Ck^{-\gamma}$.

La comparativa gráfica entre la distribución de grado puede verse en la imagen continuación:

2.4.2 De Bruijn

En teoría de grafos, un grafo de De Bruijn n -dimensional de m símbolos es un grafo dirigido que representa la superposición entre secuencias de símbolos. Tiene m^n nodos, consistiendo en todas las secuencias posibles de longitud n de los nodos dados. El mismo símbolo puede aparecer varias veces en una secuencia.

Si tenemos el conjunto de m símbolos S tal que $S := \{s_1, \dots, s_m\}$, entonces el set de nodos es $V = S^n$:

$$V = \{(s_1, \dots, s_1, s_1), (s_1, \dots, s_1, s_2), \dots, (s_1, \dots, s_1, s_m), (s_1, \dots, s_2, s_1), \dots, (s_m, \dots, s_m)\}$$

Si uno de los nodos puede ser expresado como otro nodo simplemente permutando todos sus símbolos un lugar a la izquierda y añadiendo dicho símbolo al final del nodo, entonces, el nodo resultante tiene una arista dirigida al nodo en cuestión. El sets de aristas es, por tanto:

$$E = \{((v_1, v_2, \dots, v_n), (v_2, \dots, v_n, s_i)) : i = 1, \dots, m\}$$

Los grafos de De Bruijn tienen algunas propiedades interesantes:

- Si $n = 1$, la condición de que dos nodos forman una arista se tiene siempre, por lo que se tendrán m^2 aristas que conectan todos los nodos dos a dos.
- Cada nodo tiene m aristas entrantes y m aristas salientes.
- Cada grado n -dimensional de De Bruijn es el dígrafo-linea del $(n - 1)$ -dimensional de De Bruijn para el mismo set de símbolos.
- Todo grafo de De Bruijn es euleriano y hamiltoniano.

Sin embargo, lo más importante son sus aplicaciones. Los grafos de De Bruijn binarios pueden ser dibujados de tal manera que describan objetos de la teoría de sistemas dinámicos, como un atractor de Lorenz¹².

¹²Un atractor de Lorenz es un sistema dinámico **determinista** tridimensional no lineal derivado de las ecuaciones simplificadas de rollos de convección que se producen en las ecuaciones dinámicas de la atmósfera terrestre. Para ciertos valores de las constantes $a, b, c \in \mathbb{R}$ el

C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-07-23 13-57-

Existen analogías más rigurosas. Un grafo de De Bruijn (n, m) se corresponde con un mapeo de Bernoulli¹³ $x \rightarrow mx \mod 1$, de tal manera que las trayectorias de dicho sistema dinámico corresponden a los caminos en un mapa de De Bruijn, donde la correspondencia viene dada por la aplicación del mapeo para cada $x \in [0, 1) \cap \mathbb{R}$ al nodo equivalente a los n primeros dígitos de la representación de x en base m .

Hay, además, usos directos de los grafos de De Bruijn. Algunas redes toipo reja son topologías de De Bruijn, se utilizan en protocolos de *Tablas de hash-distribuidas* y en bioinformática en la secuenciación de un genoma.

2.5 Neo4j

2.5.1 Neo4j SandBox

Neo4j no tiene herramientas asociadas que permitan simular grafos en el sentido en que se puede hacer tanto en networkx como en igraph. Sin embargo, además de siempre poder realizar en Cypher el código para cualquier grafo que queramos crear, Neo4j tiene un repositorio de grafos que llaman *SandBox*, de libre acceso y cuyo objetivo principal es la prueba y aprendizaje en Neo4j. La lista de grafos y su descripción incluidos en la plataforma en esta versión¹⁴ es la siguiente:

sistema exhiba un comportamiento caótico, por lo que es realmente estudiado. Las ecuaciones son las siguientes:

$$\begin{cases} \frac{dx}{dt} = a(y - x) \\ \frac{dy}{dt} = x(b - z) - y \\ \frac{dz}{dt} = xy - cz \end{cases}$$

¹³Un mapa de Bernoulli es un sistema dinámico ergódico, definido por la aplicación:

$$d : [0, 1) \rightarrow [0, 1)^\infty$$

$$x \rightarrow (x_0, x_1, x_2, \dots)$$

que produce la regla:

$$\begin{cases} x_0 = x \\ \forall n \geq 0, x_{n+1} = (2x_n) \mod 1 \end{cases}$$

¹⁴Versión 2 del SandBox.

- Graph Algorithms, *aprenda cómo analizar grafos con datos en Neo4j con ejemplos simples.*
- Recommendations, *genere recomendaciones personalizadas en tiempo real a partir de las reviews de un dataset de películas.*
- Crime Investifation, *explore las conexiones en los datos del crimen utilizando el modelo POLE (Person, Object, Location, Event) en un dataset público de Manchester, UK.*
- Bloom Visual Discovery, *aprenda Neo4j con un dataset de fraude y Neo4j Bloom, una aplicación para la exploración gráfica para interactuar visualmente con los datos.*
- Twitter, *permita el acceso a su cuenta de Twitter y este Sandbox le permitirá realizar un grafo con su red de Twitter, incluyendo personas y tweets.*
- ICIJ's Panama Papers, *ICIJ construyó esta base de datos y guía de políticos, criminales y demás industria de esconder su dinero.*
- TrumpWorld, *usando el dataset provisto por Buzzfeed, explore las conexiones dentro y fuera de la Administración Trump.*
- Blank Sandbox, *comience con Neo4j con una pizarra en blanco, cree su propio grafo.*
- ICIJ's Paradise Papers, *explore los últimos datos de ICIJ de las entidades off-shore de políticos, celebrities y otros individuos de renta alta.*
- Russian Twitter Trolls, *explore los datos publicados por NBC News alrededor de su investigación de los trolls de Rusia en las elecciones de 2016 de EEUU.*
- Legis-Graph, *el congreso de EEUU modelado como un grafo: votos, miembros y más.*
- Spreadsheets Grapher, *cargue datos directamente de las Spreadsheets de Google en Neo4j.*
- Network & IT Managment, *análisis de dependencia y de causa raíz a través de grafos para ahorrar tiempo y dinero.*

2.5.2 Integraciones de Neo4j

En general, las bases de datos arriba mencionadas no nos servirán en nuestro desarrollo, por lo que deberemos tratar de simular los grafos deseados de alguna manera. Como se ha propuesto anteriormente, pueden ser creados directamente a través de Cypher, pero puede resultar más fácil y conveniente integrar Neo4j con `igraph` y `networkx` para trabajar con las funcionalidades de las tres herramientas simultáneamente, con todo el potencial que esto tiene, tanto para la simulación de grafos como para el resto de campos del análisis de redes.

Aquí se dan algunos códigos posibles para transformar un grafo legible en una herramienta en otro utilizable por otra.

Via directa Neo4j - networkx

En este apartado se tratará de encontrar un método para leer un grafo de Neo4j en `networkx` eficientemente.

Sin entrar en muchos más detalles, la siguiente es una manera en que, a partir de un grafo de Neo4j, se devuelven listas donde cada una representa un nodo distinto y un conjunto de relaciones para dicho nodo¹⁵.

¹⁵Propuesto en http://www.solasistim.net/posts/neo4j_to_networkx/


```

# this version expects a collection of
# rels in the variable 'rels'
# But, this version doesn't handle
# dangling references
def rs2graph_v2(rs):
    graph = networkx.MultiDiGraph()

    for record in rs:
        node = record['n2']
        if not node:
            raise Exception('row should have a node')

        print("adding node")
        nx_properties = {}
        nx_properties.update(node.properties)
        nx_properties['labels'] =
            list(node.labels)
        graph.add_node(node.id, **nx_properties)

        relationship_list = record['rels']
        for relationship in relationship_list:
            print("adding edge")
            graph.add_edge(
                relationship.start,
                relationship.end,
                key=relationship.type,
                **relationship.properties
            )
    return graph

```

Via directa networkx - Neo4j

Existe una librería llamada *neonx* cuyo objetivo principal es convertir un grafo de networkx en su equivalente para Neo4j, aunque tiene otras funcionalidades y se basa en la conversión por pasos del formato del grafo. En primer lugar, el grafo se convierte a un string de Geoff, luego a un JSON que pueda leer el servidor de Neo4j, y así de sencillo sería su uso en caso de que queramos,

por ejemplo, codificar fechas además de los tipos usuales¹⁶:

```
import neonx

## CONVERT NETWORKX GRAPH TO GEOFF STRING
# "links_to" is the relationship name between nodes
data = neonx.get_geoff(graph, "links_to")

## ENCODE to JSON
import json
import datetime
class DateEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, datetime.date):
            return o.strftime('%Y-%m-%d')
        return json.JSONEncoder.default(self, o)

data=neonx.get_geoff(graph,"links_to",DateEncoder())

## UPLOAD GRAPH TO NEO4J
results = neonx.write_to_neo(
    "http://localhost:7474/db/data/",graph, 'links_to')
```

Networkx + Neo4j via API

Existe, también, una librería para Python creada por un desarrollador inglés¹⁷ que consigue que los algoritmos de Neo4j estén disponibles para trabajar en networkx directamente. La librería se llama *networkx-neo4j*¹⁸ y se encuentra disponible para el público bajo una licencia permisiva Apache 2.0, aunque el desarrollador menciona que algunos algoritmos están por *traducir*.

Networkx - igraph

En unas sencillas líneas de Python podemos *traducir* un grafo de igraph a

¹⁶En la documentación de *neonx*, disponible en <https://neonx.readthedocs.io/en/latest/index.html>, se menciona que las propiedades de nodos y aristas deben ser codificables a JSON para poder usar la librería directamente.

¹⁷Mark Needham, *mneedham* en GitHub. <https://markneedham.com/blog/>

¹⁸Toda la documentación acerca de esta librería puede verse aquí: <https://github.com/neo4j-graph-analytics/networkx-neo4j>

networkx y viceversa. Puede verse aquí el código propuesto¹⁹.

De igraph a networkx:

```
G = nx.DiGraph()
names = g.vs['name']
G.add_nodes_from(names)
G.add_edges_from([(names[x[0]],
                    (names[x[1]])) for x in g.get_edgelist()])
```

De networkx a igraph:

```
g = igraph.Graph(directed=True)
g.add_vertices(G.nodes())
g.add_edges(G.edges())
```

Se puede, igualmente, realizar la conversión a través de la escritura del grafo en un archivo para después leerlo, como se propone²⁰ a través de:

Caso networkx - igraph

```
import networkx as nx
import igraph as ig

Gnx = nx.path_graph(4) # Create a random NX graph

# Export NX graph to file
nx.write_graphml(G, 'graph.graphml')

# Create new IG graph from file
Gix = ig.read('graph.graphml', format="graphml")
```

¹⁹Ebrahim Byagowi en <https://stackoverflow.com/questions/23235964/interface-between-networkx-and-igraph>

²⁰Julien Deneuille, *diije*, en GitHub <https://gist.github.com/diije/193ec57d7e1347db91c52699c4f08ccd>

Part II

Medidas de centralidad

En esta parte se describirá distintos métodos de medir la centralidad o *importancia de un nodo* que está disponible en cada software de manera nativa.

3 Definición y computación de las medidas

3.1 En igraph

Las siguientes definiciones y algoritmos tratan de computar cierta noción de centralidad para un nodo (y en algunas ocasiones de una arista o relación). Además, a partir de ellas (también disponibles, en general, para igraph) de,ps generalizar inmediatamente y obtener medidas de centralización de un grafo completo, que extienden las definiciones de las medidas aquí descritas al conjunto global del grafo.

3.1.1 Burt's constraint [IG]

La constraint o *restricción* de Burt es una medida no exactamente de centralidad, sino que calcula cómo de restringido está un elemento por sus vecinos. En ese sentido, cuanto más alto sea el valor constraint, menos exposición al exterior del círculo de vecinos tendrá el nodo. Eso es muy utilizado en cadenas de información, donde el nodo con alto constraint tiende a no estar expuesto a información de fuera o a intercambiarla, mientras los denominados *brokers* (intermediarios), al traspasar información constantemente, tienen unos constraints limitados.

Formalmente, para un nodo i y un conjunto vecinal $V(j)$:

$$C(i) = \sum_{j \in V(i)} \left[(p_{ij} + \sum_q p_{iq} p_{qj})^2 \right], \quad i \neq q \neq j,$$

donde los p_{ij} están definidos como la fuerza proporcional de atadura:

$$p_{ij} = \frac{a_{ij} + a_{ji}}{\sum_{k \in V(i)} a_{ik} a_{ki}}$$

Para los nodos aislados, el valor de la constraint no está definida.

3.1.2 Grado máximo [IG]

Esta medida, tan sencilla, indica el grado (degree) máximo que un nodo tiene en la red, i.e.:

$$\text{grado_maximo}(G) = \max\{\text{degree}(j) : j \in G\}$$

3.1.3 Strength de un nodo [IG]

La *strength* o fuerza de un nodo i se define como la suma de los pesos de las conexiones con nodos vecinos. En un nodo sin pesos es exactamente el degree.

$$\text{strength}(i) = \sum_{j \in V(i)} w(j)$$

3.1.4 Kleinberg's Hub Scores [IG]

Comenzando por la definición formal del los hub-scores, estos se definen como sigue, siendo \mathbf{x} el vector de scores, que son el autovector asociado al mayor valor propio de la matriz:

$$\mathbf{A}\mathbf{A}^T$$

donde \mathbf{A} es la matriz de adyacencia del grafo.

El algoritmo, pensado inicialmente para rankear páginas web, pero desde un comienzo de carácter universal en una red; trata de identificar dos tipos de nodos distintos en una red: **authorities**, páginas relevantes que son las que *queremos* encontrar al realizar una búsqueda y **hubs**. Para encontrar dichas autoridades es complicado hacerlo en un algoritmo text-based porque, como reporta el autor, al buscar “*motor de búsqueda*” deseamos que nos encuentre páginas como *Yahoo!* o *Google*; cuando dicha cadena no aparece una sola vez en estas páginas.

La idea es, pues, aprovecharse de que otras muchas páginas tienen hiperenlaces a estas autoridades simplemente por su condición como tales. Estas otras páginas son los llamados hubs. El algoritmo en sí buscará detectar autoridades y hubs. Autoridades que, en la práctica, se colocarán primeras en los resultados de la búsqueda tratando de conjugar relevancia y popularidad.

El algoritmo parte de un subgrafo reducido S_σ , cercano al tema del string σ en cuestión que será el input del método. A partir de aquí, con una búsqueda mediante un algoritmo puramente endógeno, *text-based*, se obtienen las páginas más relevantes (en el sentido de cercanas) para dicho input.

Algorithm 1 Subgrafo($\sigma, \varepsilon, t, d$)

σ : string de la query.

ε : un motor de búsqueda *text-based*.

$t, d \in \mathbb{R}$: dos constantes.

Sea R_σ los t resultados-top de ε para σ .

Fijamos $S_\sigma := R_\sigma$.

Para cada página $p \in R_\sigma$:

 Sea $\Gamma^+(p)$ todas las páginas a las que dirige p .

 Sea $\Gamma^-(p)$ todas las páginas que se dirigen a p .

 Añadimos todas las páginas $\Gamma^+(p)$ a S_σ .

 Si $|\Gamma^-(p)| \leq d$:

 Añadimos todas las páginas de $\Gamma^-(p)$ a S_σ .

 Else:

 Se añade un set arbitrario de $\Gamma^-(p)$ de tamaño d a S_σ .

Fin.

Return S_σ .

Una vez que se obtienen estos resultados, el algoritmo recoge todas las páginas con enlaces salientes hasta un determinado grado (se sigue buscando en las páginas que salen de estas nuevas páginas), y se recogen en dos subgrupos para cada página p , $\Gamma^+(p)$ y $\Gamma^-(p)$, salientes y entrantes, respectivamente. Se añaden al subgrafo S_σ y se repite hasta que no existan hiperenlaces de menor grado que el deseado.

Tras la explicación algo informal de un algoritmo que ciertamente se ha simplificado, sigue la especificación formal:

Una vez tenemos el subgrafo S_σ , se eliminan todas las relaciones (o hiperenlaces) entre un nodo y sí mismo (de un dominio a sí mismo), resultando el grafo G_σ .

Partimos de este grafo G_σ para encontrar tanto *hubs* como *autoridades*. Partimos de la idea de que el sistema de relación es *de refuerzo mutuo*: un buen *hub* debe apuntar a muchas *autoridades*. Mientras una buena *autoridad* deberá ser apuntada por muchos *hubs*.

Para encontrar este tipo de páginas se usará un algoritmo iterativo que mantiene y actualiza dos pesos normalizados (hubs y autoridad) para cada página. Con la idea del párrafo anterior presente, se definen dos operaciones de actualización \mathcal{I} , y \mathcal{O} , dados dos conjuntos de pesos $\{x^p\}$, $\{y^p\}$ para autoridades y hubs respectivamente:

Algorithm 2 Obtención de x^* e y^*

Iteramos (G, k) .

G : una colección de n páginas enlazadas.

$k \in \mathbb{N}$: una constante.

Denotamos por z al vector $(1, 1, 1, \dots, 1) \in \mathbb{R}^n$.

Sea $x_0 := z$.

Sea $y_0 := z$.

Para cada $i = 1, 2, \dots, k$:

Se aplica la operación \mathcal{I} a (x_{i-1}, y_{i-1}) , obteniendo un nuevo x'_i .

Se aplica la operación \mathcal{O} a (x_{i-1}, y_{i-1}) , obteniendo un nuevo y'_i .

Normalizamos x'_i para obtener x_i .

Normalizamos y'_i para obtener y_i .

Fin

Se devuelve (x_k, y_k)

$$\mathcal{I}: x^p \leftarrow \sum_{q:(q,p) \in E} y^q$$

$$\mathcal{O}: y^p \leftarrow \sum_{q:(q,p) \in E} x^q$$

Representamos el conjunto $\{x^p\}$ como un vector x con una coordenada para cada página G_σ y de manera análoga para y .

Con unas comprobaciones de álgebra lineal lo suficientemente básicas, aunque algo tediosas y que no aportan gran cosa al entendimiento del algoritmo se demuestra que:

$$\begin{cases} \lim_{k \rightarrow \infty} x_k = x(\lambda_m^i) \\ \lim_{k \rightarrow \infty} y_k = y(\lambda_m^{ii}) \end{cases}$$

Siendo $x(\lambda_m^i)$ el autovector asociado a λ_m^i , el mayor autovalor de la matriz $\mathbf{A}\mathbf{A}^T$. Análogamente, $y(\lambda_m^{ii})$ será el autovector asociado a λ_m^{ii} , el mayor autovalor de la matriz $\mathbf{A}^T\mathbf{A}$.

3.1.5 Kleinberg's Authority Scores [IG]

[Véase explicación anterior]

Es una medida análoga a la anterior, el cambio en la definición es simplemente un reorden de la matriz de adyacencia con su traspuesta. De nuevo, \mathbf{x} ,

scores, es el autovector asociado al mayor valor propio de la matriz:

$$\mathbf{x} = \mathbf{A}^T \mathbf{A}$$

3.1.6 Métodos de estimación de centralidad [IG]

Para grafos extensos, donde el tiempo de computación puede ser demasiado grande, existen algunas estimaciones para ciertas medidas de centralidad ya comentadas que precisen de menos recursos:

1. **Closeness centrality:** se añade un parámetro l que define la longitud máxima que se probará en un camino. Así, si un camino llega a una longitud l y no ha llegado a su destino, se toma como su longitud exactamente l . Es inmediato, por tanto, que $Closeness \geq Closeness_estimated$.
2. **Betweenness centrality:** de nuevo se añade un parámetro l con la misma función que el anterior. Así, igraph solo considera aquellos caminos atraviesen el nodo que tengan una longitud menor que la dada. De nuevo es inmediato que $Betweenness \geq Betweenness_centrality$.

3.2 En networkx

3.2.1 VoteRank [NX]

El denominado voterank es un método para obtener aquellos nodos influyentes, en el sentido de ser aquellos con mayor centralidad. Se basa en la idea de que, en el mundo real, si una persona A apoya a otra persona B, la capacidad de apoyo de esa persona A se ve mermada.

En el desarrollo del algoritmo cada nodo u está relacionado con una tupla (s_u, va_u) , donde el primer elemento se refiere al número de votos obtenidos de los vecinos de u y, el segundo, la capacidad de voto del mismo nodo. El objetivo es obtener aquellos r nodos con mayor centralidad. Así, el algoritmo se describe en cinco pasos:

Este algoritmo, en concreto, obtiene grandes resultados en comparación a otros similares.

3.2.2 Closeness (Current-Flow based) [NX]

Partiendo de la idea de la centralidad closeness, se escoge un modelo diferente de propagación de la información: en lugar de considerar únicamente los caminos

Algorithm 3 VoteRank

Paso 1: Se inicializa el algoritmo. Todos los nodos son $(0, 1)$.

Paso 2: Se vota. Los nodos votan por sus vecinos al mismo tiempo que ellos son votados. Después de votar, se calcula el score de cada nodo. Nótese que el score del nodo escogido se actualiza a 0 para evitar volver a ser escogido. Por ejemplo, si el nodo v_0 tiene tres vecinos, v_1, v_2 y v_3 . El nodo v_0 vota cada uno con va_{v_0} y obtiene un score $s_{v_0} = va_{v_1} + va_{v_2} + va_{v_3}$.

Paso 3: Selección. Acorde a los votos del Paso 2, se selecciona el nodo v_{max} que obtiene el mayor número de votos. Desde ese instante en que está escogido, $va_{v_{max}} = 0$.

Paso 4: Actualización. Se debilitan las habilidades de voto de aquellos nodos que votaron a v_{max} en el paso 2, siendo su nueva habilidad para votar en el paso i : $v_u^i = va_u^{i-1} - f$, donde f es un factor decreciente entre 0 y 1. f se suele tomar simplemente como $f = \frac{1}{k}$ con k el grado medio (degree) de la red.

Paso 5: Se repiten los pasos de 2 a 4 hasta que obtenemos los r nodos que deseamos.

más cortos, se asume que la información es propagada **eficientemente** como en una corriente eléctrica.

Así, se define la closeness centrality (current-flow based) de un nodo s como la apicación desde el conjunto de nodos V :

$$c_{CC}(s) : V \rightarrow \mathbb{R}_{>0}$$
$$c_{CC}(s) = \frac{n_C}{\sum_{t \neq s} p_{st}(s) - p_{st}(t)} \quad \forall s \in V$$

con $n_C = n - 1$ una constante normalizadora y la diferencia $p_{st}(s) - p_{st}(t)$ que corresponde a la *resistencia efectiva* y que puede ser interpretada aquí como el correspondiente a la distancia.²¹

3.2.3 Betweenness (current-flow based) [NX]

[Véase explicación anterior]

Con la misma idea, la centralidad betweenness para un nodo v se define como la aplicación:

$$c_{CB} : V \rightarrow \mathbb{R} \geq 0$$

²¹Aunque no se derivan de la misma forma, $c_I(s)^{-1} = nC_{ss}^I + tr(C^I) - \frac{2}{n}$, la centralidad de información, con $C^I = (L + J)^{-1}$ toma exactamente el mismo valor que la centralidad closeness basada en el current-flow.

$$c_{CB}(v) = \frac{1}{n_B} \sum_{s,t \in V} \tau_{st}(v) \quad \forall v \in V$$

donde $n_B = (n-1)(n-2)$.

Para explicar el τ_{st} debemos considerar que, en redes eléctricas, lo análogo a la fracción de los caminos más cortos atravesando un nodo o una arista es la fracción de una unidad st-corriente fluyendo a través de dicho nodo (o arista). Dada un suministro b , se define el *throughput* de un nodo $v \in V$ como:

$$\tau(v) = \frac{1}{2} \left(-|b(v)| + \sum_{e:v \in e} |x(\vec{e})| \right)$$

donde el término $-|b(v)|$ da cuenta de que solo los nodos interiores se consideran en la centralidad betweenness. Para incluir el nodo inicial y final, debemos cambiar el signo por uno positivo. Sea entonces τ_{st} el *throughput* de una st-corriente.

Una st-corriente es la corriente que define un st-supply ($b_{st}(v)$):

$$b_{st}(v) = \begin{cases} 1 & v = s \\ -1 & v = t \\ 0 & \text{otro caso} \end{cases}$$

3.2.4 Communicability Betweenness [NX]

La definición de esta centralidad se basa en dos posibles formas de entender la betweenness. En un extremo: la betweenness definida sólo considerando la información de los caminos más cortos entre dos nodos. En el otro extremo, se pueden considerar todos los caminos posibles entre dos nodos, de cualquier longitud.

La idea es considerar cualquier camino de manera que obtengamos la mayor información posible, pero introduciendo un factor de escala que consiga que caminos más largos tengan menor importancia.

Para describir el algoritmo, partimos de la comunicabilidad entre dos pares de nodos. Sea P_{pq} el camino más corto entre p y q y W_{pq} el número de caminos distintos que los conecta. Definimos P_{pq}^s el número de caminos más cortos de longitud s entre p y q y W_{pq}^k el número de caminos que conecta ambos nodos de

longitud $k > s$. Podemos considerar la cantidad:

$$G_{pq} = \frac{1}{s!} P_{pq}^{(s)} + \sum_{k>s} \frac{1}{k!} W_{pq}^{(k)}$$

como una medida de *comucabilidad* entre los nodos p y q . Esta expresión, reescrita en términos de la matriz de adyacencia y deshaciendo el desarrollo en serie de la exponencial:

$$G_{pq} = \sum_{k=0}^{\infty} \frac{(A^k)_{pq}}{k!} = (e^A)_{pq}$$

Denominamos también G_{prq} la correspondiente suma de los pesos donde solo consideramos los caminos que implican al nodo r . Así, podemos definir finalmente la *communicability betweenness centrality* (CBC) de un nodo r , ω_r :

$$\omega_r = \frac{1}{C} \sum_p \sum_q \frac{G_{prq}}{G_{pq}} \quad p \neq q, p \neq r, q \neq r$$

donde, además, $C = (n-1)^2 - (n-1)$ es un factor de normalización que es el valor del número de términos en la suma. Computacionalmente usaremos la expresión (solo considerando redes con $(e^A)_{pq} \neq 0$):

$$\omega_r = \frac{1}{C} \sum_p \sum_q \frac{(e^A)_{pq} - (e^{A+E(r)})_{pq}}{(e^A)_{pq}} \quad p \neq q, p \neq r, q \neq r$$

Para explicar esta nueva expresión, comenzamos con el grafo completo $G = (V, E)$ y eliminamos todas las aristas en las que se conecte el nodo r , quedando $G(r) = (V, E')$. La matriz de adyacencia de $G(r)$ podrá escribirse como $A + E(r)$ donde $E(r)$ será idénticamente nula excepto en la fila y en la columna r , donde tendrá -1 allí donde A tenga $+1$.

Directamente de esto, podemos escribir:

$$G_{prq} = (e^A)_{pq} - (e^{A+E(r)})_{pq}$$

3.2.5 Load Centrality [NX]

Esta medida de centralidad es la fracción de todos los caminos más cortos que pasan a través de un nodo²². Es algo distinto a la centralidad *betweenness* una

²²Existe una versión para las aristas análoga a la de los nodos.

vez nos acercamos a la hora de su cálculo, ya que parte de que sigue una ley potencial: $P_L(l) \sim l^{-\delta}$, y se aproxima $\delta \approx 2.2$.

Mientras la betweenness es *prohibitiva* en términos computacionales, el *load*, l_i , sigue una fórmula:

$$\frac{l_i}{\sum_j l_j} \sim \frac{1}{N^{1-\beta_i\beta}}$$

con $\beta = 0.80$.

Se escala el *load* total $\sum_j l_j \sim N^2 \log N$, debido a que hay N^2 pares de nodos en la red y la suma del *load* contribuido por cada par de nodos es igual a la distancia de entre los dos vértices, $\propto \log N$. De aquí, l_i viene dado por:

$$l_i \sim (N \log N)(N/i)^\beta$$

Finalmente, puede demostrarse una dependencia entre el grado (degree) de un grafo y su *load*:

$$l \sim k^{(\gamma-1)/(\delta-1)}$$

Siendo γ el exponente de la ley potencial que sigue el grado de un grafo: $P_D(k) \sim k^{-\gamma}$, que numéricamente $\gamma \in (2, 3)$.

3.2.6 Percolation (*filtración*) [NX]

Esta medida de centralidad proviene de ámbitos donde existe filtración o *percolation*, como por ejemplo la infección de virus informáticos (o no informáticos) en una red social de individuos. Con esta nueva medida se conseguirá cuantificar el impacto relativo de los nodos basados en su conectividad topológica, así como su estado de filtración.

Se define la *percolation centrality* (PC) de un nodo v **en un instante** t , como sigue:

$$PC^t(v) = \frac{1}{N-2} \sum_{s \neq v \neq r} \frac{\sigma_{s,r}(v)}{\sigma_{s,r}} \frac{x_s^t}{(\sum x_i^t) - x_v^t}$$

donde $\sigma_{s,r}(v)$ es el número de caminos más cortos entre el nodo de origen s y de destino r que pasa a través de v . Por otro lado, x_r^t , se define como el estado de filtración en el nodo r en un instante t . Dicho nivel o estado de filtración $x_r^t \in [0, 1]$ que, a mayor valor, más filtración y viceversa.

Existen algunos casos particulares:

1. Si $x_i^t = 0 \ \forall i \in I \Rightarrow PC(v) = 0 \ \forall v \in V$

Algorithm 4 Algoritmo de segundo orden

Hasta la llegada del camino aleatorio al nodo i :

Se escoge un vecino j de Γ_i aleatoriamente

Se computa d_j , grado de j

Se genera un número aleatorio $p \in [0, 1]$

Si $p \leq \frac{d_i}{d_j}$:

El camino continua en j

Else:

El camino permanece en i

Si es la primera visita del camino a i :

Se crea el array Ξ_i

Else:

Se computa el tiempo r desde la última visita

Añadimos r a Ξ_i

Si $|\Xi_i| \geq 3$, entonces:

Se computa la desviación estandar:

$$\sigma_i(N) = \sqrt{\frac{1}{N-1} \sum_{k=1}^N \Xi_i(k)^2 - \left[\frac{1}{N-1} \sum_{k=1}^N \Xi_i(k) \right]^2}$$

2. Si solo un nodo tiene percolation positiva (en cuyo caso $\frac{x_s^t}{(\sum x_i^t) - x_v^t} = 1$)
ó todos tienen una filtración completa ($\frac{x_s^t}{(\sum x_i^t) - x_v^t} = \frac{1}{N-1}$), ambos
casos llevan inmediatamente²³ a la betweenness centrality.

3.2.7 Second order centrality [NX]

La centralidad de segundo orden de un nodo v se basa en la idea de que un nodo central en la topología será visitado con mayor recurrencia que nodos no-centrales. El punto en que la centralidad de segundo orden mejora la centralidad a través de caminos aleatorios de Newman es lo siguiente, como él mismo explicita en su trabajo: un nodo no-central pero con un grado alto tiende a tener una puntuación alta, en tanto que la correlación es entre puntuación y grado y no entre puntuación y centralidad, lo que crea un efecto no deseado siempre que la red no sea homogénea en la distribución del grado.

El algoritmo de la centralidad de segundo orden, para evitar dicho efecto, se basa en la desviación típica de los tiempos de visita al nodo de un camino aleatorio perpetuo en la red. Formalmente:

²³En el caso en que solo un nodo tiene filtración positiva deben ponderarse todos los escenarios posibles y realizar la media de los resultados obtenidos para hallar exactamente la betweenness centrality.

Sin embargo, en `networkx` no se utiliza el algoritmo que incluye la simulación de caminos aleatorios, sino que se utiliza una expresión analítica para la desviación típica de un nodo j :

$$\sigma(j) = \sqrt{2 \sum_{i \in S} M(i, j) - |S|(|S| + 1)}$$

Siendo S el espacio de estados finito de la cadena de Markov en tiempo **discreto** (DTMC) que describe el proceso, i.e. S es el conjunto de nodos de la red.

Por otra parte, $M(i, j) = \mathbb{E}\{\tau(j)|X_0 = i\}$, es decir, el número de transiciones que, de media, necesita la cadena para ir del estado inicial i al estado final j .

Dicha matriz $M(i, j)$ es obtenible pues:

$M_j = (I - Q_j)^{-1}\mathbf{1}$, con dicha matriz Q_j la matriz de transición P a la que se reemplaza la columna j por un vector de ceros.

Se trata de un algoritmo pesado, con complejidad $\mathcal{O}(n^3)$, lo que lo hace adecuado solo para redes simples.

3.2.8 Reaching centrality [NX]

Se dará la explicación del algoritmo tanto para la centralidad local de un nodo dado, como para la centralidad global, que también está implementada en `networkx`.

La definición de la reaching centrality busca tres objetivos simultáneamente: ausencia de parámetros o de una métrica *a priori* en la definición; adecuación a grafos no dirigidos y fácilmente extendible a grafos con peso y dirigidos y, finalmente; debe ser útil para generar un *layout* del grafo.

La reaching-centralidad $C_R(i)$ de un nodo i se define como la proporción de todos los nodos de la red que pueden ser alcanzados desde i :

$$C_R(i) = \frac{\#\{\text{nodos alcanzables } i\}}{N}$$

Sea ahora C_R^{max} como el máximo valor de la centralidad local para el conjunto $i \in I$, conjunto de nodos del grafo y se define la GRC, *global reaching centrality*:

$$GRC = \frac{\sum_{i \in I} [C_R^{max} - C_R(i)]}{N - 1} \leq 1$$

En general, para grafos dirigidos y con pesos²⁴:

²⁴Es inmediato comprobar que para grafos dirigidos pero sin peso $\omega_i = 1 \forall i$.

$$C'_R(i) = \frac{1}{N-1} \sum_{j: 0 < d^{out}(i,j) < \infty} \frac{\sum_{k=1}^{d^{out}(i,j)} \omega_i^{(k)}(j)}{d^{out}(i,j)}$$

3.2.9 Subgraph centrality [NX]

La noción de esta centralidad descansa en la idea de caracterizar la participación de cada nodo en cada subgrafo del grafo. A mayor tamaño del subgrafo, mayor peso en dicha caracterización.

Sea G un grafo de orden N . Se define el espectro de G como el conjunto de autovectores de la matriz de adyacencia de dicho grafo. A partir de él podemos entender la densidad espectral como la densidad de los autovalores de su matriz de adyacencia, que puede relacionarse directamente con las propiedades topológicas del grafo a través de los momentos espectrales. En concreto, el número de caminos cerrados de longitud k que comienzan y terminan en el nodo i de la red se denota por $\mu_k(i)$ que puede definirse simplemente como la entrada i -ésima de la diagonal de la k -ésima potencia de la matriz de adyacencia:

$$\mu_k(i) = (\mathbf{A}^k)_{ii}$$

A partir de aquí, definimos la centralidad de subgrafos del nodo i como:

$$SC(i) = \sum_{k=0}^{\infty} \frac{\mu_k(i)}{k!}$$

Sea ahora $v = (v_1, \dots, v_N)$ una base ortonormal de \mathbb{R}^N compuesta por los autovectores de \mathbf{A} asociados a los valores propios $\lambda_1, \dots, \lambda_N$. Para un nodo dado $i \in V$, la centralidad-subgrafo (SC) puede expresarse como:

$$SC(i) = \sum_{j=1}^N (v_j^i)^2 e^{\lambda_i}$$

3.2.10 Índice de Estrada [NX]

La idea de esta definición será la de dar un valor o índice topológico a la *compactidad* en 3D, primeramente pensada para sistemas moleculares.

La aproximación teórica es la siguiente: en primer lugar, se considera un grafo G . Ahora se identifican las aristas de G como los nodos de un nuevo grafo definiendo que dos nodos de éste serán adyacentes si y sólo si las correspondientes

aristas de G inciden al mismo nodo. Denotamos a este nuevo nodo como $L(G)$, nodo de primera línea de G . Si repetimos el proceso, $L^2(G)$ será el nodo de segunda línea de G .

Consecuentemente, si consideramos la matriz de adyacencia de $L^3(G)$ con los nodos ponderados por una función correspondiente a los ángulos diedros resulta una matriz $\mathbf{M} = \mathbf{A}(L^3(G)) + \Delta$, donde Δ es una matriz diagonal cuyo n -ésimo término es el coseno del n -ésimo ángulo diedro de G .

A partir de los momentos espectrales ya explicados, se define el índice I_E , *Índice de Estrada*, como:

$$I = \sum_i e^{\lambda_i}$$

Siendo $\lambda_i \in sp(M)$ los autovalores de la matriz M .

3.3 En Neo4j

3.3.1 Article Rank [N4J]

Esta medida de centralidad parte o es una variante del PageRank específico para el análisis de redes de citas. El desarrollo es muy similar, y lo mejor es comparar directamente las ecuaciones que especifican el algoritmo:

PageRank:

$$PR(A) = 1 - d + d \sum_{i=1}^n \frac{PR(P_i)}{O(P_i)}$$

con d y $1 - d$ nuestros α y β respectivamente. $O(P_i)$ es el outdegree de la página P_i , una de las n que tienen link a A .

ArticleRank:

$$AR(A) = 1 - d + \bar{NR} \cdot d \sum_{i=1}^n \frac{AR(P_i)}{\bar{NR} \cdot NR(P_i)}$$

Como vemos, la única diferencia entre ambas ecuaciones es la agregación de una ponderación por la media del valor de NR para todos los nodos de las red. Este arreglo se realiza con el fin de evitar un sesgo en la medida de centralidad: un nodo (un paper) con muy pocas relaciones salientes (referencias), tendrá una contribución mayor (pero irreal) que un nodo (paper) de las mismas características pero con muy pocas referencias.

3.4 Compartidas

3.4.1 Pagerank [IG, N4J]

Llamado así por ser el algoritmo utilizado por Google inicialmente para escoger el orden de las páginas en su motor de búsqueda. Se basa en que un nodo es importante si está ligado a otros importantes y a nodos no importantes o si está altamente conectado. La definición, un poco más compleja, es la siguiente:

Sea $A = (a_{ij})$ la matriz adyacente del grafo dirigido. La centralidad-pagerank, x_i de un nodo i es:

$$x_i := \alpha \sum_k \frac{a_{k,i}}{d_k} x_k + \beta$$

donde α y β son constantes y d_k es el degree-saliente (outdegree) del nodo k . Si dicho grado no existiera, se define $d_k = 1$.

De forma matricial tenemos:

$$\mathbf{x} = \alpha \mathbf{x} \mathbf{D}^{-1} \mathbf{A} + \beta$$

donde β ahora es un vector cuyos elementos son una constante dada. Por su parte, \mathbf{D}^{-1} es una matriz diagonal con el i -ésimo elemento de la diagonal $(\mathbf{D})_{ii} = \frac{1}{d_i}$. Por ello, como puede comprobarse, el PageRank está determinado por un componente endógeno y otro exógeno, independiente este de la estructura de la red. Suele computarse de la siguiente manera:

$$\mathbf{x} = \beta (\mathbf{I} - \alpha \mathbf{D}^{-1} \mathbf{A})^{-1}$$

dem:

$$\mathbf{x} = \alpha \mathbf{x} \mathbf{D}^{-1} \mathbf{A} + \beta \iff \mathbf{x}^{-1} \mathbf{x} = \alpha \mathbf{x}^{-1} \mathbf{x} \mathbf{D}^{-1} \mathbf{A} + \mathbf{x}^{-1} \beta$$

$$(\mathbf{x}^{-1} \beta)^{-1} = (\mathbf{I} - \alpha \mathbf{D} \mathbf{A})^{-1} \iff \mathbf{x} = \beta (\mathbf{I} - \alpha \mathbf{D}^{-1} \mathbf{A})^{-1}$$

Además, el valor de la variable d suele ser 0.85, pues representa la probabilidad de que “un navegante continúe pulsando links al navegar por Internet en vez de escribir una url directamente en la barra de direcciones o pulsar uno de sus marcadores y es un valor establecido por Google. Por lo tanto, la probabilidad de

que el usuario deje de pulsar links y navegue directamente a otra web aleatoria es $\beta = 1 - d$.” Una alternativa al PageRank es el algoritmo Hits propuesto por Jon Kleinberg, que se verá más adelante.

En igrph se pueden especificar los vértices para los que se quiere calcular dicha centralidad y hay varias versiones de este algoritmo, pero en esencia son el mismo método.

3.4.2 Degree [IG, NX, N4J]

Simplemente el grado de cada nodo, es decir, el número de relaciones que tiene con los demás nodos.

$$\text{degree}(i) = \sum_{i \neq j} r_{ij}$$

Siendo r_{ij} una arista que une i con j , pudiendo calcular para el caso direccional, el out-degree y el in-degree (relaciones salientes y entrantes respectivamente).

3.4.3 Closeness Centrality [IG, NX, N4J]

El *closeness* o cercanía (Bavelas, 1950) es una medida básica de centralidad que trata de medir la facilidad con la que se pueden alcanzar otros vértices a partir del nodo en cuestión. Se define de la siguiente manera, dado un nodo x :

$$\text{Closeness}(x) := \frac{N - 1}{\sum_{y \neq x} d(y, x)}$$

Si el grafo no es conexo, en aquellos casos en que x no sea alcanzable desde y , se cambia la distancia por el número de vértices totales del grafo.

3.4.4 Betweenness centrality [IG, NX, N4J]

La centralidad *betweenness* de un nodo es el número de caminos que pasan a través de él. Si hay dos caminos posibles entre dos grafos, el valor de los mismos en el cálculo es ponderado por el número de caminos. En general:

$$\text{Betweenness}(x) := \#\{\text{path}(y, z) : x \in \text{path}(y, z), x \neq y \neq z \neq x\}$$

Análogamente se puede definir este tipo de centralidad para las aristas.

3.4.5 Centralidad a través de los autovectores [IG, NX, N4J]

Esta medida de importancia de un nodo en la red asigna scores relativos a cada nodo en la red basado en el principio de que las conexiones con nodos de alto scores contribuyen más al score del nodo en cuestión que las mismas conexiones con nodos con puntuaciones más bajas. Para ello, se calcula simplemente el autovector asociado al autovalor λ_N positivo más alto de la matriz de adyacencia A .

La centralidad relativa x_i se define formalmente como:

$$x_i = \frac{1}{\lambda} \sum_{j \in V(i)} x_j = \frac{1}{\lambda} \sum_{j \in G} a_{i,j} x_j$$

Con un pequeño arreglo, puede verse que se halla simplemente calculando el autovector:

$$\mathbf{Ax} = \lambda \mathbf{x}$$

En general, podría existir solución para muchos valores de λ , pero el teorema de Perron-Frobenius establece que solo el mayor de los autovalores garantiza que \mathbf{x} sea positivo como necesitamos.

Para hallar el mayor autovalor numéricamente se utiliza el *power method*.

De hecho, el pagerank de Google es una variante de esta medida de centralidad.²⁵

3.4.6 Harmonic centrality [NX, N4J]

Similar a la closeness, se define la centralidad armónica como:

$$\sum_{y \neq x} \frac{1}{d(x, y)} = \sum_{d(y, x) < \infty, y \neq x} \frac{1}{d(y, x)}$$

donde $d(x, y)^{-1}$ se toma como 0.

El motivo de esta nueva definición es que el problema con la closeness habitual radica en la presencia de nodos inalcanzables cuya distancia, Marchiori y Latora, proponen reemplazar con la *harmonic mean of all distances*. Sin embargo, en caso de que haya muchos pares de nodos no alcanzables, esto puede llevar a una mala definición de la centralidad, por lo que se opta por la fórmula descrita al comienzo del párrafo.

²⁵En networkx podemos encontrar dos formas distintas de calcular esta centralidad: una *estándar* y otra basada en numpy.

3.5 Otras medidas de centralidad

3.5.1 ClusterRank

Disponible de manera no-nativa en R a través de la librería *centiserve*. Matemáticamente, el score ClusterRank s_i de un nodo i se define como:

$$s_i = f(c_i) \sum_{j \in \tau_i} (k_{out}^j + 1)$$

donde el término $f(c_i)$ es el efecto del clustering en i y el término $+1$ resulta de la contribución del propio j . Aquí, $f(c_i) = 10^{-c_i}$.

3.5.2 Rumor-source

Esta medida de centralidad se basa en la probabilidad de que un nodo sea fuente de un rumor. A mayor centralidad, mayor probabilidad o cercanía a la fuente del rumor. La definición es la siguiente:

Sea $v \in V$ un nodo del grafo $G = (V, E)$, sea $T \subset G$ el árbol de primera-amplitud o *first-breadth* de v con respecto G . Así, la centralidad-rumor de un nodo v se define:

$$R(v, G) = \frac{|V|!}{\prod_{w \in V} T_w^v}$$

donde V es el conjunto de nodos del grafo G y T_w^v es el tamaño del subárbol de G con raíz en w que *señala* el nodo v .

Dicha expresión es, justamente, la probabilidad de que un nodo sea fuente del rumor. Además, se dirá que la *fuentes del rumor* es aquel nodo con una centralidad-rumor más alta.

3.6 Librería CINNA en R

Esta librería cuyo nombre completo es *Central Informative Nodes in Network Analysis* contiene 43 formas diferentes de medir la centralidad de un grafo. Muchas de ellas ya se han visto con anterioridad pero otras no. La lista completa es la siguiente:

```
## [1] "subgraph centrality scores"
## [2] "Topological Coefficient"
## [3] "Average Distance"
## [4] "Barycenter Centrality"
## [5] "BottleNeck Centrality"
```

```

## [6] "Centroid value"
## [7] "Closeness Centrality (Freeman)"
## [8] "ClusterRank"
## [9] "Decay Centrality"
## [10] "Degree Centrality"
## [11] "Diffusion Degree"
## [12] "DMNC – Density of Maximum Neighborhood Component"
## [13] "Eccentricity Centrality"
## [14] "eigenvector centralities"
## [15] "K–core Decomposition"
## [16] "Geodesic K–Path Centrality"
## [17] "Katz Centrality (Katz Status Index)"
## [18] "Kleinberg’s authority centrality scores"
## [19] "Kleinberg’s hub centrality scores"
## [20] "clustering coefficient"
## [21] "Lin Centrality"
## [22] "Lobby Index (Centrality)"
## [23] "Markov Centrality"
## [24] "Radiality Centrality"
## [25] "Shortest–Paths Betweenness Centrality"
## [26] "Current–Flow Closeness Centrality"
## [27] "Closeness centrality (Latora)"
## [28] "Communicability Betweenness Centrality"
## [29] "Community Centrality"
## [30] "Cross–Clique Connectivity"
## [31] "Entropy Centrality"
## [32] "EPC – Edge Percolated Component"
## [33] "Laplacian Centrality"
## [34] "Leverage Centrality"
## [35] "MNC – Maximum Neighborhood Component"
## [36] "Hubbell Index"
## [37] "Semi Local Centrality"
## [38] "Closeness Vitality"
## [39] "Residual Closeness Centrality"
## [40] "Stress Centrality"
## [41] "Load Centrality"

```

[42] "Flow Betweenness Centrality"
[43] "Information Centrality"

4 Cómo escoger una medida de centralidad

Desde un punto de vista teórico, no existe consenso entre los investigadores a la hora de escoger una entre las diferentes medidas de centralidad.

Además, muchas de estas medidas tienen correlaciones negativas, en el sentido en que alta centralidad con unas conlleva baja centralidad en la otra y viceversa: es el caso de *topological coefficient* y la descrita *subgraph centrality*.

Desde un punto de vista empírico podemos acercarnos a una solución a través del análisis de componentes principales, *PCA*. Para ello, debemos utilizar las medidas de centralidad como variables: las medidas de centralidad que están correladas con los componentes principales son más importantes en la identificación de los nodos centrales. Por ello, *deberá*²⁶ escogerse la medida de centralidad que tiene una mayor contribución via *PCA*, porque será aquella que sea capaz de extraer mayor información de los nodos centrales.

²⁶Siempre que sea posible deberá escogerse la centralidad buscada de manera teórica. Por ejemplo, en la propagación de un vídeo viral no tiene sentido considerar la centralidad *pagerank* por delante de la *centralidad-rumor*, independientemente del resultado del *PCA*.

Part III

Detección de comunidades

5 Descripción de los algoritmos

En esta primera sección se describirán los algoritmos disponibles para cada software. Muchos de ellos serán compartidos por lo que tendrán su epígrafe correspondiente en la subsección conjunta, como podría ser “*igraph + networkx*”.

5.1 En igraph

5.1.1 Algoritmo basado en física estadística (statistical mechanics) [IG]

J. Reichardt y Stefan Bornholdt (1). Parten del Hamiltoniano del sistema que debe ayudar a separar comunidades en base a cuatro aspectos fundamentales de las mismas: a.) debe premiarse aristas internos entre nodos del mismo grupo, b.) penalizar aristas faltantes entre nodos del mismo grupo, c.) penalizar aristas internas entre distintos grupos y d.) premiar aristas faltantes entre nodos de odistinto grupo. Esto lleva a la siguiente función:

$$\begin{aligned}\mathcal{H}(\{\sigma\}) = & -\sum_{i \neq j} a_{ij} A_{ij} \delta(\sigma_i, \sigma_j) + \sum_{i \neq j} b_{ij} (1 - A_{ij}) \delta(\sigma_i, \sigma_j) + \\ & + \sum_{i \neq j} c_{ij} A_{ij} (1 - \delta(\sigma_i, \sigma_j)) - \sum_{i \neq j} d_{ij} (1 - A_{ij}) (1 - \delta(\sigma_i, \sigma_j))\end{aligned}$$

La primera parte de la ecuación hace referencia a los enlaces internos efectivos, la segunda a los no-enlaces internos, la tercera a los enlaces externos y la cuarta a los no-enlaces externos; siendo A la matriz de adyacencia y σ_i denota el “spin state” o índice de grupo.

A partir de aquí proponen que los grupos deberán ser aquellos que hagan al sistema llegar al estado fundamental (ground state), es decir, el estado de energía más bajo posible en un vidrio de espín o spin glass infinito (un sistema magnético donde el acoplamiento entre los momentos magnéticos de los distintos átomos es aleatorio).

Parámetros en python:

- `weights = None`
- `spins = 25`, número de *spins* a usar, que limita el número máximo de comunidades.
- `parupdate=False`, se escoge si los *spins* se actualizan sincronizadamente o no.
- `start_temp=1`, temperatura inicial.
- `stop_temp=0.01`, temperatura de parada.
- `cool_fact=0.99`, factor de enfriamiento.
- `update_rule="config"`, especifica el modelo de simulación. “config”, un grafo aleatorio con el mismo grado en los nodos que el input ó “simple”, un grafo aleatorio con el mismo número de aristas.
- `gamma=1`, el argumento gamma del algoritmo que especifica el equilibrio entre la importancia de aristas presentes o faltantes en una comunidad.
- `implementation="orig"`, igraph tiene dos formas de implementación. “orig” es más rápida, mientras que “neg” permite pesos negativos.
- `lambda=1`, el argumento del algoritmo que especifica el equilibrio entre la importancia de aristas con pesos negativos presentes o faltantes en una comunidad. Sólo si se usa “neg”.

5.1.2 Algoritmo basado en los autovectores de las matrices [IG]

Desarrollado por M. Newman (2). Parte de la matriz de Laplace (una transformación de la matriz de adyacencia) o matriz de admisión. El objetivo es minimizar R (cut size), que definimos como el número de aristas que salen fuera de un grupo para ir a otro:

$$R = \frac{1}{2} \sum_{i,j \text{ d.g.}} A_{ij}$$

Refiriendo i, j d.g. a los elementos de distintos grupos. R , a su vez, puede ser expresada en términos de la matriz de admisión y el vector índice $S = (S_1, S_2, \dots, S_N)$ que para cada variable. Si toma el valor 1 si el nodo pertenece al grupo 1 y -1 si pertenece al segundo grupo:

$$s_i = \begin{cases} 1 & i \in \text{grupo 1} \\ -1 & i \in \text{grupo 2} \end{cases}$$

Tal que minimizar R será equivalente a minimizar:

$$R = \frac{1}{4} s^T L s$$

Después de llegar a una solución trivial $s=1$ (si solo existe un grupo, no existen conexiones entre grupos), se impone una nueva restricción y se obtiene la solución del problema de optimización planteado:

$$s_i = \begin{cases} 1 & v_i^{(2)} \geq 0 \\ -1 & v_i^{(2)} < 0 \end{cases}$$

Donde $v_i^{(2)}$ es la i -ésima componente del autovector asociado al segundo autovalor más pequeño de L , también llamado “Fiedler Vector”.

Parámetros en python:

- `n=-1`, número deseado de comunidades. Si es negativo, hará tantas particiones como pueda el algoritmo.
- `arpack_options=None`, `ARPACKOptions` es un objeto usado para refinar el cálculo de autovectores
- `weights=None`

5.1.3 Método Infomap [IG]

El método Infomap, desarrollado por M. Rosvall y C. T. Bergstrom. Se basa en la *map equation* a la que llegaron con el fin de tratar de simplificar los datos en redes de gran tamaño. Es un método basado en los flujos existentes entre nodos que consigue hallar el límite teórico de cómo de conciso se podría especificar un camino en la red usando una partición dada de la estructura.

Con el fin de encontrar una partición óptima de la red, es suficiente calcular dicho límite teórico para diferentes particiones de la misma y escoger aquella que devuelva el la mínima longitud descriptiva.

Se define esa cota mínima como $L(M)$, la ya citada *map equation*:

$$L(M) = q_{\curvearrowright} H(\mathcal{Q}) + \sum_{i=1}^m p_{\cup}^i H(\mathcal{P}^i)$$

Para comprender esta ecuación debe recordarse el Teorema de Codificación de Fuentes de Shannon, que implica que, dados n *codewords* que sirven de descripción a n estados de una variable aleatoria X , que toma probabilidades p_i para cada estado, la longitud media de un *codeword* no puede ser menor que la entropía de dicha variable aleatoria: $H(X) = -\sum_1^n p_i \log(p_i)$. Se podrá estimar la probabilidad por la frecuencia de uso de cada *codeword*. Así, $H(\mathcal{Q})$ refiere a la longitud media ponderada por la frecuencia de los *codewords* en el libro índice y $H(\mathcal{P}^i)$ es lo propio para los *codewords* del módulo i -ésimo. Siendo $q_{i\curvearrowright}$ es la probabilidad de salida del módulo i -ésimo, si sumamos en i obtenemos q_{\curvearrowright} , el ratio al que es usado el libro índice. Siendo p_{α} la probabilidad de visitar el nodo α , el libro del módulo i -ésimo se usa a un ratio: $p_{\cup}^i = \sum_{\alpha \in i} p_{\alpha} + q_{i\curvearrowright}$.

Hay algunas diferencias en el método entre redes direccionales y no direccionales. Como ejemplo, en las primeras se utiliza el método de las potencias para hallar la frecuencia de visita para cada nodo en el estado estacionario. Sin embargo, en ambos métodos se trata de minimizar la ecuación $L(M)$. Los autores exponen que este problema es resoluble con cualquier algoritmo numérico que optimice una función objetivo cuya variable de control sea la partición de la red.

En concreto, se utilizará un algoritmo de búsqueda voraz (rápido pero poco preciso), otorgándole propiedades estocásticas con el fin de conseguir una mayor precisión. Básicamente, el algoritmo es el siguiente: nodos vecinos se unen en módulos que, seguidamente, se unen en supermódulos y así sucesivamente.

Primero, cada nodo se asigna a su propio módulo. Después, en un orden secuencial aleatorio, cada nodo se traslada al módulo vecino que aporte un mayor descenso al valor de $L(M)$. Si este traslado no reportara una diferencia negativa en $L(M)$, se dejaría en el módulo en el que está. Este procedimiento es repetido, siempre en un orden secuencial aleatorio, hasta que ningún movimiento consiga hacer descender el valor de la *map equation*. En ese momento, la red es reconstruida con los módulos anteriores formando nodos en esta nueva red.

Esta reconstrucción jerárquica se repite hasta que la ecuación no pueda tomar un valor menor. Finalmente, se devuelve aquella red que consiguió la mínima $L(M)$.

Parámetros de la función en python:

- `edge_weights=None`,
- `vertex_weights=None`,
- `trials=10`.

Trials es el número de intentos de dividir la red. Además, la función devuelve un atributo extra que almacena la longitud del código determinada por el algoritmo.

5.1.4 Algoritmo de la modularidad óptima [IG]

A través del GNU Linear Programming Kit se resuelve un problema de programación entera para encontrar la modularidad óptima y su correspondiente estructura de comunidad. Por ello sólo es recomendable para redes pequeñas (<100 nodos).

5.1.5 Algoritmo walktrap [IG]

Basado en el algoritmo planteado por P. Pons y M. Latapy. La idea central es que los caminos aleatorios de un grafo tienden a quedarse *atrapados* dentro de las partes más densamente conectadas, correspondientes a comunidades.

Consideraremos un camino aleatorio discreto en un grafo G . En cada instante el camino pasa de un nodo a otro aleatoria y uniformemente. La secuencia de estados visitados es una cadena de Markov con matriz de transición P tal que $P_{ij} = \frac{A_{ij}}{d(i)}$. Se cumple lógicamente la propiedad markoviana:

$$P(X_t = j | X_{t-1} = i, X_{t-2}, X_{t-3}, \dots) = P(X_t = j | X_{t-1} = i) := P_{ij}$$

Para obtener la necesaria comparabilidad entre vertices, se define una distancia r entre nodos que permitirá recoger las similitudes entre nodos, capturando la estructura de comunidad en el grafo:

$$r_{ij} = \sqrt{\sum_{k=1}^n \frac{(P_{ik}^t - P_{jk}^t)^2}{d(k)}} = \|D^{-1/2}P_{i\cdot}^t - D^{-1/2}P_{j\cdot}^t\|$$

donde la norma utilizada es la usual (euclidiana) en \mathbb{R}^n .

Generalizamos ahora la distancia entre nodos a la distancia entre comunidades. Definimos primeramente la probabilidad de ir de la comunidad C al nodo j en t pasos:

$$P_{Cj}^t = \frac{1}{|C|} \sum_{i \in C} P_{ij}^t$$

Obteniendo así un vector de probabilidad P_C^t , que nos permite definir:

$$r_{C_1 C_2} = \|D^{-1/2}P_{C_1\cdot}^t - D^{-1/2}P_{C_2\cdot}^t\| = \sqrt{\sum_{k=1}^n \frac{(P_{C_1 k}^t - P_{C_2 k}^t)^2}{d(k)}}$$

Una vez que tenemos definida la distancia, podemos demostrar que la matriz P y la distancia están relacionadas de la siguiente manera:

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

donde λ_{α} y v_{α} son los autovalores y autovectores respectivamente de la matriz de transición P . Esta distancia, además, es fácilmente generalizable introduciendo los pesos del grafo.

El algoritmo es el siguiente:

1. Se toma una partición $\mathcal{P}_1 = \{\{v\}, v \in V\}$ del grafo con n comunidades reducidas a un nodo. Computamos las distancias entre todos los vertices adyacentes. Después, esta partición evoluciona repitiendo, a cada paso k :
2. Se escogen dos comunidades C_1 y C_2 en \mathcal{P}_k , de acuerdo al criterio basado en la distancia que se detalla más abajo.
3. Se funden ambas comunidades en una nueva $C_3 = C_1 \cup C_2$ y creamos una nueva partición: $\mathcal{P}_{k+1} = (\mathcal{P}_k \setminus \{C_1, C_2\}) \cup \{C_3\}$.
4. Se actualizan las distancias entre comunidades (sólo necesario para las adyacentes).

Después de $n - 1$ pasos, el algoritmo termina y obtenemos $\mathcal{P}_n = \{V\}$. Cada paso define una partición del grafo en comunidades, por lo que se obtiene una estructura jerárquica llamada **dendrograma**: un árbol donde las ojas corresponden a los nodos y en cada rama se unen a otros nodos creando comunidades cada vez más grandes.

El punto clave del algoritmo es la manera en que se escoge qué comunidades deben unirse. Este proceso se realizará de acuerdo al método de Ward. En cada paso k se unirán aquellas comunidades que minimizar la media de las distancias al cuadrado entre cada vértice y su comunidad.

$$\sigma_k = \frac{1}{n} \sum_{C \in \mathcal{P}_k} \sum_{i \in C} r_{iC}^2$$

Para una mejor computación se opta por obtener la variación $\nabla\sigma(C_1, C_2)$ que sólo dependerá de los nodos de ambas comunidades y no de todas las demás. Finalmente, se unirán las dos comunidades con el menor valor de $\nabla\sigma$.

Para encontrar la mejor partición, consideramos que es aquella que maximiza:

$$Q(\mathcal{P}) = \sum_{C \in \mathcal{P}} e_C - a_C^2$$

Con e_C la fracción de aristas dentro de la comunidad C y la fracción de aristas a_C que se dirigen a dicha comunidad. Sin embargo, dependiendo del objetivo podría considerarse otra evaluación, también a través del dendrograma.

Parámetros en python:

- weights=None
- steps=None

Devuelve una tupla con la lista de fusiones y los resultados de modularidad en cada una de ellas.

5.2 En networkx

5.2.1 Bipartición de Kernighan-Lin [NX]

Se trata de un algoritmo heurístico, aunque puede obtenerse una gran solu-

ción (probablemente la óptima) con rapidez.

Sea $G = (V, E)$ un grafo no direccional con conjunto de nodos V y conjunto de aristas E . La intención es buscar una partición de V tal que los dos subconjuntos disjuntos A y B sean de aproximadamente el mismo tamaño, de manera que se minimice la suma T de pesos de aristas que cruzan de A a B . Si el grafo no tuviera pesos, se minimizaría el número de aristas. El algoritmo mantiene y mejora una partición; en cada paso usa un algoritmo voraz (greedy) para unir nodos en A con nodos en B , de tal manera que trasladar los nodos de un lado de la partición a otra consiga mejorar la misma.

Con mayor detalle, para todo $a \in A$, sea I_a el coste interno de a , es decir, la suma de los costes de las aristas entre a y otros nodos en A . Sea ahora E_a , el coste externo de a . Es decir, la suma de costes de aristas entre a y nodos en B . Análogamente definimos I_b , $E_b \forall b \in B$.

Así, podemos entender $D_s = E_s - I_s$ como la diferencia entre el coste externo e interno de s . Por ello, si a y b se intercambian, la reducción total será:

$$T_{old} - T_{new} = D_a + D_b - 2c_{a,b}$$

donde $c_{a,b}$ es el coste de la posible arista entre a y b . El algoritmo es el siguiente:

1. Se determina una partición inicial equilibrada de nodos en dos conjuntos, A y B . Creamos gv , av y bv como listas vacías.
2. Se computan los valores D y se toman aquellos elementos a y b de los conjuntos que maximicen $g = T_{old} - T_{new}$ y se guardan en av y bv respectivamente, así como g en gv .
3. Se eliminan a y b del algoritmo y actualizamos los valores de D con $A_i = A_{i-1} - a$ y $B_i = B_{i-1} - b$. Encontrar k que maximice $gmax = \sum_{l=1}^k gv[l]$. Si $gmax > 0$, se intercambia $av[1], av[2], \dots, av[k]$ con $bv[1], bv[2], \dots, bv[k]$ hasta que $gmax \leq 0$.

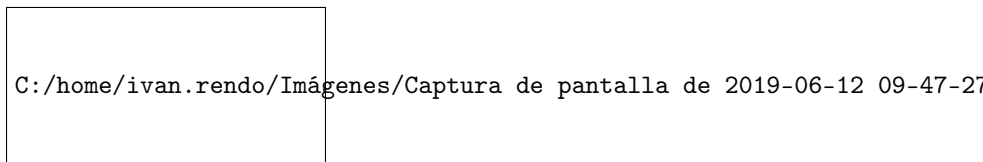
Se devuelve $G(V, E)$.

5.2.2 K-clique comunidades a través de filtración [NX]

Este algoritmo está especialmente diseñado para un cierto tipo de grafos que

son los k -cliques, para ello lo definiremos con anterioridad.

Sea G un grafo $G = (N, A)$. Decimos que G tiene un clique de tamaño k (ó k -clique) si $\exists G' = (N', A') \subset G : N' \subset N, |N'| = k$ y $A' = N' \times N'$. Es decir, todos sus vértices están conectados entre ellos. En la siguiente figura, $\{1, 2, 5\} \subset G$ forman una k -clique.



Así, con este método trataremos de encontrar k -clique-comunidades como uniones de estas k -cliques anteriores.

El algoritmo en primer lugar extrae todas las cliques de la red que no son parte de otras mayores. La diferencia entre una clique y una k -clique es que la primera no puede ser parte de otra mayor. Una vez localizadas las cliques, podemos crear una matriz simétrica de adyacencia de cliques donde sus coeficientes representen el número de nodos comunes entre las cliques. Como se muestra en el primer paso de la imagen que sigue este texto.

El segundo paso es sencillo, para hallar una k -clique-comunidad, debemos determinar en primer lugar k . A partir de ahí, eliminamos en la matriz anteriormente definida los elementos de la diagonal más pequeños que k otorgándoles 0 (y 1 a las que se mantienen) eliminando aquellas cliques que fueran *uniones* entre comunidades. Además, se eliminaran las relaciones entre cliques que no superen $k - 1$ conexiones entre nodos de ambas. En la matriz se pondrá 1 si existe relación y 0 si no, una vez eliminadas las *débiles*. Así, llegamos directamente a una matriz que asocia un grafo separado en comunidades, como puede verse en el ejemplo.

Función en networkx:

`k_clique_communities(G, k, cliques=None)`

- k – tamaño mínimo de la clique
- cliques (list) – cliques precomputadas

5.2.3 Fluid communities (asincronizado) [NX]

Este algoritmo está basado en la metodología de la propagación, teniendo una precisión semejante a otros métodos en la detección de comunidades y siendo, además, el primer método basado en propagación capaz de identificar un número variable de comunidades.

El algoritmo se basa en la idea de introducir un número determinado de fluidos (comunidades) en un ambiente no homogéneo donde los fluidos se expandirán y *empujarán* unos a otros influidos por la topología hasta alcanzar un estado estacionario.

Sea $G = (V, E)$ un grafo con la notación usual. El algoritmo inicializa k comunidades $\mathcal{C} = \{c_1, \dots, c_k\}$, donde lógicamente $0 < k \leq |V|$. Cada comunidad k comienza en un nodo aleatorio $v \in V$. Para continuar, definimos la **densidad** de una comunidad como la inversa del número de nodos componiendo dicha comunidad.

$$d(c) = \frac{1}{|v \in c|} \in (0, 1]$$

El algoritmo opera por superpasos. En cada uno de ellos, se itera sobre todos los nodos en V en orden aleatorio. Cuando la asignación de los nodos a comunidades no cambia en dos superpasos se para el algoritmo, puesto que ha convergido.

La regla de actualización para un nodo específico v será, formalmente:

$$\mathcal{C}' = \underset{w \in \{v, \Gamma(v)\}}{\in \mathcal{C}} \sum d(c) \delta(c(w), c)$$

$$\delta(c(w), c) = \begin{cases} 1 & c(w) = c \\ 0 & c(w) \neq c \end{cases}$$

Siendo v el nodo a actualizar, \mathcal{C}'_v los candidatos a ser la nueva comunidad de v , $\Gamma(v)$ los vecinos de v , $d(c)$ la densidad de la comunidad c , $c(w)$ la comunidad del nodo w y $\delta(c(w), c)$ son las deltas de Kronecker.

Si se da el caso en que \mathcal{C}' esté formado por más de una comunidad, hay dos opciones: primero, que la actual comunidad del nodo a actualizar se encuentre en el conjunto, i.e. $c(v) \in \mathcal{C}'$, en cuyo caso no se actualiza la comunidad; o que ocurra lo contrario, donde se decidirá a qué comunidad pertenece el nodo de forma uniformemente aleatoria.

Es decir, se devuelve en la actualización la comunidad o comunidades con

una densidad máxima agregada dentro de la propia red *vecinal* de v .

Función en networkx:

`asyn_fluidc($G, k, max_iter=100, seed=None$)`

- k – número de comunidades deseadas.
- `max_iter` – número de iteraciones permitidas. 15 por defecto.
- `seed` – indicador de la semilla de aleatorización.

5.3 En Neo4j

5.3.1 Componentes conectados [N4J]

Basado en el algoritmo de Galler y Fischer que utiliza una estructura de árbol y que puede ser aplicado a cualquier problema en que sea necesario identificar clases de equivalencia, una vez que se define la relación de equivalencia.

En el caso de los grafos, el algoritmo simplemente detecta conjuntos de nodos conectados en un grafo no-direccional donde cada nodo es alcanzable por cualquier otro nodo en el mismo conjunto. Los componentes del grafo pueden ser computados utilizando o bien BFS (Búsqueda en anchura), se comienza en la raíz dada y se exploran todos los vecinos del nodo para, a continuación, explorar los respectivos vecinos adyacentes a cada uno de los vecinos originales ó DFS (Búsqueda en profundidad), expandir todos y cada uno de los nodos que va localizando de forma recurrente en un camino concreto.

5.3.2 Componentes conectados fuertemente [N4J]

Este algoritmo sirve para encontrar conjuntos de nodos conectados en un grafo direccional donde cada nodo es alcanzable en ambas direcciones desde cualquier otro nodo en el mismo conjunto. Se suele utilizar en estudios preliminares.

Sea G el grafo a explorar. El algoritmo inicia en algún nodo aleatorio de G y escoge, de nuevo aleatoriamente, un arista que parta de ese nodo v_1 y lleve a un nuevo nodo v_2 . Se continua de esa manera: en cada paso se selecciona un arista no-explorado de un nodo alcanzado y se atraviesa ese arista para alcanzar un nodo v_i que puede haber sido o no descubierto con anterioridad. Si se llega a no tener aristas por explorar, se escoge un nodo no-alcanzado y se comienza una nueva exploración. Eventualmente se recorrerán todas las aristas del grafo.

5.3.3 Conteo de triángulos / Coeficiente cluster [N4J]

Esta técnica se basa en el conteo de triángulos y suele utilizarse tanto para detectar comunidades como para medir la cohesión de las mismas o la estabilidad del grafo a través, además, de los coeficientes de clustering de un nodo: **local**, la probabilidad de que sus vecinos estén también conectados; o **global**, la usma normalizada de los coeficientes locales. El objetivo final es conseguir una tabla como esta:

Nombre	Triángulos	Coeficiente
A	1	1
B	1	1
C	2	0.6
D	2	0.6
E	3	0.3
F	0	0

Donde se observa dos características diferentes de cada elemento. Aunque el elemento *E* es el más *popular* (si entendemos popular como perteneces a más triángulos), son los individuos *A* y *B* tienden a ser los mejores vinculando a sus vecinos entre sí.

Para contar los triángulos a los que pertenece cada nodo, que es lo realmente necesario, se utiliza una serie de algoritmos provistos por T. Schank y D. Wagner.

- **Algoritmo *node-iterator*:** este algoritmo itera sobre todos los nodos y prueba para cada par de vecinos si están conectados por un arista. La cota máxima de tiempo computacional es $\mathcal{O}(nd_{max}^2)$ siendo $d_{max}(G)$ el máximo grado (degree) alcanzado por un nodo en el grafo G .
- **Algoritmo *ayz* y *listing-ayz*:** este algoritmo fue creado por Alon, Yuster y Zwick. Este algoritmo divide el conjunto de nodos en aquellos con bajo y alto grado: $V_{low} = \{v \in V : d(v) \leq \beta\}$, y $V_{high} = V \setminus V_{low}$, donde $\beta = m^{\frac{\gamma-1}{\gamma+1}}$. El método estándar *node-iterator* se utiliza en V_{low} y *matrix-multiplication* en V_{high} . El tiempo computacional será $\mathcal{O}(m^{2\gamma/(\gamma+1)})$.

- **Algoritmo *node-iterator-core*.** Sea un k -core el mayor subgrafo de nodos con un grado mínimo de k . El *core number* $c(v)$ de un nodo v será el máximo k de todos los cores a los que pertenece. El algoritmo toma un nodo con el mínimo grado, computa sus triángulos de la misma manera que el *node-iterator* y después elimina el nodo del grafo. El tiempo computacional es $\mathcal{O}(nc^2)$.
- **Algoritmo *edge-iterator*.** Este algoritmo itera sobre las aristas y compara la estructura de los datos de adyacencia con los nodos que conecta el arista. Para un arista $\{u, v\}$, los nodos $\{u, v, w\}$ forman un triángulo si y sólo si el nodo v está presente en ambas matrices de adyacencia. El tiempo computacional será $\mathcal{O}(md_{max})$.
- **Algoritmo *forward*.** Es un refinamiento del anterior. En lugar de utilizar la matriz de adyacencia conteniendo todos los vecinos del nodo v , se utiliza una matriz de estructura dinámica $A(v)$.

5.3.4 Triadas equilibradas [N4J]

Este algoritmo sirve para evaluar el equilibrio estructural en el grafo. Basado en la Teoría del Equilibrio (*Balanced Theory*) de Fritz Heider.

El algoritmo consigue contar el número de triadas equilibradas y no-equilibradas. En grafos con signos, comprobar el estado de una triada es simplemente una operación elemental con los signos del triángulo a estudiar en el grafo:

$$Triada : \begin{cases} equilibrada & sg(e_1) * sg(e_2) * sg(e_3) = 1 \\ no - equilibrada & sg(e_1) * sg(e_2) * sg(e_3) = -1 \end{cases}$$

siendo e_i las aristas (lados) del triángulo para $i = \{1, 2, 3\}$.

5.4 Algoritmos compartidos

5.4.1 Algoritmo multinivel o de Louvain [IG, N4J]

Se trata de un algoritmo *bottom up* (de abajo arriba). Se reconoce la rapidez del algoritmo fast-greedy de Clauset et al. y se decide que es un buen punto de comienzo, si bien tiende a crear supercomunidades, a obtener resultados bajos en cuestión de modularidad en comparación y, como consecuencia de lo primero, el algoritmo no es aplicable sin introducir algunos cambios en redes realmente grandes (~2 millones de nodos).

La limitación del algoritmo multinivel no es ya el tiempo computacional, sino la capacidad de almacenamiento, pues se consigue identificar comunidades con 118 millones de nodos en 152 minutos.

El algoritmo en sí tiene dos fases. En primer lugar, se asignan los diferentes nodos de la red a diferentes comunidades. Ahora se considera para cada nodo i sus vecinos j . Se computa la ganancia de modularidad de quitar a i de su propia comunidad para traspasarle a la comunidad de j . Se recoloca a i , como es lógico, en aquella comunidad de sus vecinos en la cual el aumento de modularidad sea máximo, pero positivo. Este primer paso se repite secuencialmente (el orden parece irrelevante en el resultado, pero puede ralentizar el proceso) y repetidamente para todos los nodos y termina cuando ninguno de ellos tiene mejora posible en términos de modularidad.

Gran parte de la eficiencia del proceso radica en la computación de la ganancia de modularidad de pasar un nodo i a una comunidad C :

$$\nabla Q = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right]$$

donde \sum_{in} es la suma de todos los pesos de las conexiones dentro de la comunidad C , \sum_{tot} la suma total de los pesos de conexiones en los que están presentes los nodos de la comunidad C , k_i es la suma de los pesos de las conexiones en las que está presente i ; mientras $k_{i,in}$ es la suma de los pesos de las conexiones de i a los nodos en C y m es la suma de los pesos de todas las conexiones en la red.

La segunda fase del algoritmo consiste en construir una nueva red cuyos nodos son ahora las comunidades de la primera fase; los nuevos pesos entre comunidades vienen dados por la suma de los pesos de las conexiones intercomunitarias entre los antiguos nodos. Una vez esta fase está completada, se repite de nuevo la primera fase. Los pasos se iteran hasta que no hay más cambios y se obtiene un máximo en la modularidad.

Parámetros en python:

- weights=None,
- return_levels=False.

Si return_levels=True, las comunidades en cada nivel son devueltas mediante una lista.

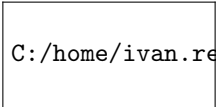
5.4.2 Algoritmo basado en la betweenness (intermediación) [IG, NX]

La centralidad betweenness es una medida de centralidad basada en los caminos más cortos. Definida formalmente como:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Donde σ_{st} es el número total de caminos más cortos desde el nodo s al nodo t y $\sigma_{st}(v)$ es el número de esos caminos que pasan por v .

En esta medida se basa el algoritmo creado por M. Girvan y M. E. J. Newman. Comienza a partir de la idea de, en lugar de tratar de construir una medida donde se muestre qué aristas son más centrales, centrarse en buscar cuáles lo son menos: esos son los aristas entre comunidades, buscando las conexiones en gris del siguiente esquema:



C:/home/ivan.rendo/Imágenes/Captura de pantalla de 2019-06-11 10-2

Para ello podemos generalizar la centralidad betweenness de Freeman antes definida a los aristas, con una analogía directa sustituyendo nodos por aristas. La idea es que, si una comunidad está diferenciada sustancialmente de otra, habrá relativamente pocos aristas que conecten nodos de ambas comunidades. Por lo tanto, por esos aristas tendrán que pasar los caminos más cortos entre dichos nodos, otorgando a los aristas intercomunitarios una alta betweenness.

El algoritmo será el siguiente:

1. Se calcula la centralidad betweenness para todos los aristas de la red.
2. Se eliminan los aristas con mayor centralidad.
3. Se recalcula la betweenness para todos los aristas afectados por la medida.
4. Se repite desde el paso 2 hasta que no quede ningún arista.

Así, se consigue separar las comunidades de la red, eliminando los aristas grises del esquema.

Parámetros de igraph:

- directed=True
- weights=None

Parámetros en networkx: girvan_newman(G[,most_valueable_edge]), el segundo argumento permite decidir cuál es el criterio a seguir para eliminar el arista más importante. Si no se especifica, será aquel con el mayor betweenness centrality.

5.4.3 Método de la propagación de etiquetas [IG, NX, N4J]

Basado en el algoritmo creado por Raghavan. La idea principal es la siguiente:

Supongamos un nodo x que tiene como vecinos a x_1, x_2, \dots, x_k y que cada nodo vecino posee una etiqueta denotando la comunidad a la que pertenece. Asumimos que el nodo x se unirá a la comunidad que tiene el máximo número de vecinos perteneciendo a ella; con desempates uniformemente aleatorios.

El método se iniciará con una etiqueta única para cada nodo y se deja que éstas se propaguen por la red. Según el proceso avanza, grupos de nodos densamente conectados alcanzan rápidamente un consenso en una única etiqueta. Cuando estos grupos homogéneos se crean, tratan de expandirse por la red tomando nuevos nodos hasta que es posible.

Al final del proceso de propagación, los nodos con las mismas etiquetas se agrupan como una comunidad.

Para evitar problemas de una iteración infinita causada por ciclos o desempates, se decide parar el proceso iterativo en aquel momento en que cada nodo

en la red tiene una etiqueta que es la predominante entre sus vecinos. Con esto se consigue una partición donde cada nodo tiene al menos tantos vecinos en su comunidad como en cualquier otra, es decir, finalmente:

$$i \in C_m \implies d_i^{C_m} \geq d_i^{C_j} \forall j$$

Siendo C_k la etiqueta k, y $d_p^{C_q}$ el número de vecinos que el nodo p tiene etiquetados como la comunidad q.

Parámetros en igraph:

- weights=None,
- initial=None,
- fixed=None.

En *initial* podemos otorgar una lista conteniendo las etiquetas iniciales para los nodos. En *fixed* se pondrá una lista de booleanos donde True significará que el nodo correspondiente no debe ser cambiado durante el algoritmo.

Parámetros en networkx:

Existen dos funciones distintas donde podemos escoger entre una propagación sincronizada de la etiqueta: *label_propagation_communities(G)* o asincrónica, con *asyn_lpa_communities(G[,weight,seed])*.

5.4.4 Algoritmo fastgreedy [IG, NX]

Este algoritmo es más ligero computacionalmente y está especialmente indicado para detectar comunidades en redes con un gran número de elementos. Basado en la optimización voraz (greedy) de la modularidad de la red. La modularidad se define como la fracción de aristas que se hallan dentro del grupo menos la esperanza de la fracción de aristas que se hallasen dentro del grupo si éstos se distribuyeran aleatoriamente. Hay diferentes formas de calcularla y pertenece, por definición, al conjunto $[-1, 1] \cap \mathbb{R}$.

En este caso se utilizará la siguiente:

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

Siendo v y w dos nodos. A la matriz de adyacencia, k_v el grado (degree) del nodo w , c_v la comunidad a la que pertenece v , m es el número de aristas de la red y δ refiere a las deltas de Kronecker.

El planteamiento inicial es tratar las comunidades i, j como nodos de un multigrafo. Para realizar un algoritmo eficiente, aunque el objetivo sea buscar los pares i, j que consigan maximizar ∇Q_{ij} , los autores proponen solo mantener aquellos pares que estén unidos por al menos un arista, ya que son los únicos que pueden aumentar el valor de Q .

Los elementos del algoritmo son los siguientes:

1. Una matriz dispersa conteniendo ∇Q_{ij} para cada par de comunidades con al menos un arista entre ellos.
2. Un min-max heap (montículo) H que contiene el mayor elemento de cada fila de la matriz ∇Q_{ij} junto a las etiquetas de las correspondientes comunidades.
3. Un vector con elementos a_i , la fracción de aristas que terminan la comunidad i .

Así, el algoritmo se puede describir de la siguiente manera:

1. Se calculan los valores iniciales de ∇Q_{ij} y de a_i de acuerdo a ciertas ecuaciones aproximativas. Se crea el max-heap con el elemento más grande de cada fila de la matriz ∇Q .
2. Se selecciona el máximo ∇Q_{ij} en H para unir ambas comunidades y se actualizan las matriz ∇Q , el heap H y el vector a_i .
3. Se repite el paso 2 hasta que solo quede una comunidad.

La estructura final de la red por comunidades será aquella que alcanzó el máximo valor de modularidad Q .

Parámetros de la función en `igraph`: `weights=None`, lista con los atributos de los aristas o su peso.

Parámetros en `networkx`: `weight (opc)`.

6 Comparativa tiempo computacional

Sea m el número de aristas y n el de nodos.

1. Walktrap [IG]: $\mathcal{O}(mn^2)$. Si la matriz es dispersa, $\mathcal{O}(n^2 \log n)$.
2. Kernighan-Lin bipartido [NX]: $\mathcal{O}(n^2 \log n)$.
3. K-cliques [NX]: No puede darse un tiempo en general. Se dice que es un problema no-polinomial pero depende demasiado de la estructura del input como para dar una cota cerrada. Empíricamente pueden apreciarse una serie de curvas que determinan el tiempo computacional t , donde $t = Am^{B \ln(m)}$, donde m es el número de aristas y A, B parámetros de ajuste.
4. Label Propagation (LPA): $\mathcal{O}(m)$.
5. Fluid Community Algorithm (FCA): $\mathcal{O}(m)$.
6. Union-find de Galler y Fisher: $\mathcal{O}(n + nm)$.
7. DFS de Robert Tarjan: $k_1 n + k_2 m + k_3 : k_1, k_2, k_3 \in \mathbb{R}$

Part IV

Conectividad de una red

7 Homofilia (o asortatividad)

7.1 Definición

Se define la *homofilia* o *asortatividad* de un nodo como la preferencia de este por unirse a otros que le son similares en alguna característica frente a los demás.

7.2 Medida

Hay diferentes medidas de asortatividad dependiendo en las características más relevantes que deseemos recoger. En concreto veremos tres de ellas.

7.2.1 Coeficiente de asortatividad

El coeficiente de asortatividad no es más que el coeficiente de correlación lineal de Pearson calculado para el grado entre pares de nodos conectados. Como sabemos, $r \in (-1, 1)$, donde un valor cercano a la unidad quiere decir asortatividad perfecta, uno nulo no-asortatividad y uno negativo desasortatividad. Se calcula como:

$$r = \frac{\sum_{jk} jk(e_{jk} - q_j q_k)}{\sigma_q^2}$$

El término q_k es la distribución del *grado residual*, que obtiene el número de aristas que parten del nodo sin contar aquella que conecta el par. La distribución de q_k viene dada por:

$$q_k = \frac{(k+1)p_{k+1}}{\sum_{j \geq 1} j p_j}$$

Finalmente, e_{jk} se refiere a la distribución conjunta de la probabilidad residual de los grados de ambos nodos. Esta cantidad es simétrica en un grafo no dirigido y sigue: $\sum_{jk} e_{jk} = 1$, $\sum_j e_{jk} = q_k$.

En un grafo dirigido existen cuatro métricas que surgen de extender el coeficiente de asortatividad: $r(in, in)$, $r(out, out)$, $r(in, out)$, $r(out, in)$. Siendo, en general:

$$r(\alpha, \beta) = \frac{\sum_i (j_i^\alpha - \bar{j}^\alpha)(k_i^\beta - \bar{k}^\beta)}{\sqrt{\sum_i (j_i^\alpha - \bar{j}^\alpha)^2} \sqrt{\sum_i (k_i^\beta - \bar{k}^\beta)^2}}$$

para $\alpha, \beta = \{in, out\}$.

7.2.2 Conectividad de vecino

Otra manera de capturar la correlación entre grados es examinar las propiedades de $\langle k_{nn} \rangle$, el grado medio de los vecinos de un nodo con grado k . Formalmente:

$$\langle k_{nn} \rangle = \sum_{k'} k' P(k' | k)$$

donde $P(k' | k)$ es la probabilidad condicional de que una arista de un nodo con grado k se dirija a otro con grado k' .

Si esta función es creciente, la red es asortativa y, de manera análoga, si es decreciente no lo será.

7.2.3 Asortatividad Local

En redes asortativas pueden existir nodos que sean desasortativos y viceversa. Por ello se define una medida local de asortatividad, que para grafos no dirigidos es la siguiente:

$$\rho = \frac{j(j+1)(\bar{k} - \mu_q)}{2M\sigma_q^2}$$

donde j es el grado de un nodo particular, \bar{k} es la media del grado de los vecinos y M es el número de links de la red. Análogamente puede definirse para redes dirigidas.

7.3 Computación

Estas medidas vienen de manera nativa en `igraph` y `networkx`, tanto para atributos como para nodos numéricos, y con algunas especificaciones más típicas de las funciones de estas librerías.

8 Otros

8.1 Multiplicidad

La multiplicidad está asociada sobre todo al análisis de redes sociales y se define como el número de tipos de relaciones contenidos en un enlace. Por ejemplo, dos personas que trabajen juntas y que, además, sean amigas, tendrán multiplicidad dos en el contexto en que esas relaciones estén descritas en el grafo.

La multiplicidad se asocia fundamentalmente con la fuerza de la relación, en sentido positivo.

Tanto `networkx` como `igraph` y `Neo4j` permiten trabajar con relaciones múltiples, por lo que esta medida de multiplicidad, aunque no de manera nativa con dicho nombre, es fácilmente computable para cada nodo.

8.2 Mutualidad o reciprocidad

La reciprocidad se define en términos de dos nodos i y j . Si el nodo i tiene una relación tipo w con j , diremos que existe reciprocidad si j mantiene una relación tipo w con i , para $w \in W$, conjunto de tipos de relaciones.

$$reciprocidad(i, j) = \begin{cases} 1 & \text{si } i \xrightarrow{w} j \wedge j \xrightarrow{w} i \\ 0 & \text{otro caso} \end{cases}$$

Tanto `networkx` como `igraph` tienen esta función integrada.

8.3 Network Closure

Una medida de *cierre de la red*. Se basa en la idea local de transitividad, una propiedad que se da cuando dos nodos relacionados a un mismo nodo también están relacionados entre sí. Se mide, sin embargo, cuántas (o el porcentaje) de triadas de este tipo están efectivamente *cerradas* en el grafo.

Existe una función en `networkx` y `igraph` que devuelve el censo de triadas cerradas de un grafo, además de otras para calcular la transitividad.

8.4 Propinquity

Es la variante geográfica de la homofilia explicada con anterioridad: dos elementos rienden a tener un mayor número de relaciones cuando aumenta la cercanía geográfica.

Part V

Otras propiedades

9 Descriptivas

9.1 Tamaño

El tamaño de un grafo puede referirse al número N de nodos o, menos comúnmente, al número E de aristas. $E \in [N - 1, E_{max}]$, siendo $E = N - 1$ en el caso de un árbol y $E = E_{max}$ en el caso de un grafo completo. Si el grafo es simple (no se puede repetir un mismo enlace entre dos nodos), $E_{max} = \binom{N}{2} = \frac{N(N-1)}{2}$; que será el doble en caso de que el grafo sea dirigido: $E_{max} = N(N-1)$. Si se permiten conexiones a sí mismo: $E_{max} = N^2$, y si no se restringe a un grafo simple, $E_{max} = \infty$.

Tanto el número de ejes totales como el número de nodos es fácilmente computable por networkx, igraph y Neo4j.

9.2 Densidad

La densidad de una red se refiere al ratio entre aristas *realizadas* y el número posible de las mismas en una red con N nodos. Así, por lo dicho anteriormente:

$$D = \frac{E - (N - 1)}{E_{max} - (N - 1)}$$

La densidad viene implementada en networkx y en igraph.

9.3 Grado medio

El grado k de un nodo es el número de aristas conectadas a él. Así, podemos calcular el grado medio directamente:

$$\langle k \rangle = \frac{2E}{N}$$

en el caso no direccional, y:

$$\langle k \rangle = \frac{E}{N}$$

en el caso direccional. En un grafo aleatorio $G(N, p)$ donde cada nodo se conecta a otro con probabilidad p podemos computar el valor esperado de $\langle k \rangle$ como:

$$\mathbb{E}[\langle k \rangle] = \mathbb{E}[k] = p(N - 1)$$

En este caso, ni `igraph` ni `networkx` traen implementados la función de grado medio, pero es fácilmente computable a partir del número de aristas y de nodos y no es especialmente interesante.

9.4 Camino más corto característico

También llamado *camino más corto medio* es, simplemente:

$$d_{uv}^C = \mathbb{E}[d_{uv}] \quad \forall u, v \in N$$

siendo d_{uv} el camino más corto entre u y v .

El comportamiento de dicha esperanza como una función a computar muestra un posible efecto *small-world*²⁷. Dados N nodos, si el tiempo computacional asciende a $\mathcal{O}(\ln N)$, el modelo presenta redes *small-world*. Crecimiento más rápido que el algoritmo muestra que no se producen *small-worlds*. El caso especial es $\mathcal{O}(\ln \ln N)$, conocido como un efecto *ultra-small-world*.

Podemos calcular esta propiedad a través con una función específica de `networkx` y de `igraph`.

10 De distribución

10.1 Puentes

Decimos que un nodo i es un *punte* si rellena un *agujero estructural*, proveyendo el único enlace entre otros dos nodos o clusters. También se puede incluir el caso en que el nodo i es necesario para una ruta corta donde una más larga no es factible debido a un alto riesgo de fallo en la transmisión.

10.1.1 Bridge-finding a través del algoritmo de Tarjan

El algoritmo de Tarjan de 1974 es el primero que consigue encontrar todos los puentes de un grafo en tiempo lineal. El método, formalmente, es el descrito en

²⁷El efecto *small-world* se refiere al fenómeno de que la distancia media en la red es muy pequeña comparada con el tamaño de la red.

Algorithm 5 Algoritmo de Tarjan para localización de puentes.

Se busca un *spanning forest* (subgrafo con pocos ejes que recorra todos los nodos del grafo) de G

Se crea un bosque F del tipo *rooted-forest* que parta del anterior.

Se recorre F en preorder y se numeran los nodos. Los padres en el árbol tienen números menores que sus hijos.

Para cada nodo v en preorder:

 Se computa el número de descendientes $ND(v)$ del nodo, añadiendo 1 a la suma de los descendientes de sus hijos.

 Se computa $L(v)$, la etiqueta con el valor más bajo en el preorder alcanzable desde v por un camino por el cual todas las aristas excepto la última están dentro del subárbol enraizado en v .

 Similarmemente, se computa $H(v)$, el valor más alto en el preorder alcanzable por un camino que todas excepto la última arista se mantiene dentro del subárbol enraizado en v .

 Para cada nodo w con un nodo padre v , si $L(w) = w$ y $H(w) < w + ND(w)$, entonces la arista de v a w es un puente.

Algorithm 6 Chain decomposition

Sea todo nodo del grafo G etiquetado como *no-visitado*.

Para cada nodo v en la sucesión tipo *depth-first search* con nodos etiquetados $1, \dots, n$, se recorre cada arista que no esté en el árbol DFS que incide en v y se sigue el camino de vuelta hacia la raíz T , parando en el primer nodo que esté marcado como *visitado*. Durante este recorrido, todo nodo se marca como *visitado*. Por ello, la parada se produce en un nodo que forma un camino dirigido o ciclo, comenzando en v : las cadenas o *chain*. La i -ésima cadena encontrada se refiere como c_i . $C = C_1, C_2, \dots$ es la descomposición en cadenas de G .

Así, una arista e de G será un **puente** si y sólo si e no está contenido en ninguna cadena de C .

Algoritmo 5.

Este algoritmo no está implementado de manera nativa en networkx ni en igraph.

10.1.2 Bridge-finding con descomposición en cadena

Este algoritmo es el que utiliza networkx para encontrar los puentes de una red, y se caracteriza de la manera que sigue en *Algoritmo 6*.

10.2 Structural Holes

10.2.1 Definición

Un *agujero estructural* se puede entender como un espacio (falta de conexión) entre dos individuos (nodos) que tienen fuentes complementarias a la información. En ese sentido, la teoría de los *structural holes* sugiere que ciertas posiciones en la red consiguen ventajas o desventajas para los nodos que las ocupan.

Una medida simple para comprobar la existencia de estos agujeros sería el **conteo de puentes** como explicamos con anterioridad.

10.2.2 Medida a través del tamaño efectivo

Burt introdujo la medida de la redundancia de una red con el fin de estimar qué contacto j es redundante con otros contactos del nodo i en términos informativos. Así, la redundancia de un nodo i se define:

$$Redundancy(i) = p_{iq}m_{jq}$$

donde p_{iq} es la proporción de *energía* de i gastada en mantener la conexión con q . Por otro lado, m_{jq} se calcula como la interacción de j con q dividida de la interacción más alta de j con cualquier nodo de la red. A partir de esto podemos definir el tamaño efectivo (ES) de la red del nodo i :

$$ESN(i) = \sum_j \left[1 - \sum_q p_{iq}m_{jq} \right] \quad q \neq i, j \neq i$$

Cuanto más desconectado esté el nodo de otros contactos primarios, mayor será el tamaño efectivo. $ESN(i) \in [1, N]$, donde 1 es el valor para la red que sólo provee un único enlace y N donde es totalmente no-redundante.

En este sentido, **Borgatti** simplificó la fórmula para redes sin peso (*unweighted*). Siendo las especificaciones:

$$Redundancy(i) = \frac{2t}{n}$$

$$ESN(i) = n - \frac{2t}{n}$$

donde t es el número total de enlaces y n el número de nodos de la red *egocéntrica* (se excluyen los enlaces al nodo i).

Mientras `igraph` no calcula el tamaño efectivo, `networkx` sí lo hace.

10.2.3 Medida a través de la restricción

Ya se ha explicado con anterioridad el concepto de *constraint*. Así, el *constraint* de una red es la suma de cada una de las locales:

$$c_{ij} = (p_{ij} + \sum_q p_{iq}p_{qj})^2 \quad i \neq q \neq j$$

Este indicador mide el grado en que tiempo y energía se concentran en un único cluster. Consiste en dos componentes: directo, donde un contacto consume cierta proporción de estas variables en la red, e indirecto, donde un contacto controla otros individuos que consumen tiempo y energía de la red.

Ambas librerías, *igraph* y *networkx*, tienen una función dedicada al cálculo de la *constraint*.

10.3 Fuerza del vínculo o *Tie Strength*

Se define como la combinación lineal de tiempo, intensidad empocional, intimidad y reciprocidad. Altos valores de estas fuerzas vinculantes se relacionan con homofilia, propinquity y transitividad; mientras que bajos valores se relacionan con puentes.

Esta definición, al ser algo más abstracta y *personalizable* que las anteriores, no se encuentra predefinida en *igraph* ni en *networkx*.

10.4 Eficiencia

10.4.1 Definiciones

La eficiencia de una red es la medida de cómo de eficiente intercambia información y es un concepto aplicable tanto a nivel local como global.

Podemos definir la eficiencia media $E(G)$ de un grafo G como:

$$E(G) = \frac{1}{n(n-1)} \sum_{i \neq j \in G} \frac{1}{d(i, j)}$$

donde n denota los nodos totales en una red y $d(i, j)$ el camino más corto entre los nodos i y j .

Por su parte, la eficiencia global $E_{glob}(G)$ puede definirse como:

$$E_{glob}(G) = \frac{E(G)}{E(G^{ideal})}$$

donde G^{ideal} es el grafo *ideal* de n nodos en el sentido de que todas las posibles aristas están presentes.

A un nivel local, como alternativa del coeficiente de clustering, podemos definir:

$$E_{loc}(G) = \frac{1}{n} \sum_{i \in G} E(G_i)$$

donde G_i es el subgrafo local consistente únicamente en los vecinos del nodo i , pero sin el nodo en sí mismo.

Existen funciones en `networkx` para computar estas definiciones, al igual que en `igraph`.

10.4.2 Aplicaciones

De una manera vaga, podemos entender la eficiencia como un cuantificador del comportamiento *small-world*.

Además, también puede utilizarse para determinar estructuras eficientes en términos económicos en redes con pesos (costes distintos) o sin ellos (costes unitarios).

10.4.3 Otras nociones de eficiencia en redes

Es usual encontrar en el mundo redes, en este caso nos centramos en bipartitas, que relacionan un conjunto A de elementos y otro conjunto B , de tal manera que al menos un conjunto de los dos tiene preferencias sobre el otro. Un ejemplo pueden ser las plazas de un colegio público, que -en principio y legalmente- no tiene preferencias sobre los alumnos que las ocupan pero que, sin embargo, los alumnos sí tienen una ordenación de preferencias sobre ellas.

Un ejemplo en que ambos conjuntos tienen preferencias sobre el otro podría ser el caso en que n cantantes busquen asociarse a un guitarrista entre los m existentes para formar un grupo, con las repercusiones que tiene si $n \neq m$.

Una asignación eficiente en este sentido donde las preferencias juegan un papel importante podemos definirla en el sentido Pareto, donde una asignación es eficiente si no existe otra que mejore al menos la situación de un elemento sin empeorar a los demás.

La literatura sobre esta noción de eficiencia es amplia y se escapa del objetivo central del análisis de redes, por lo que sólo se dará el nombre de los algoritmos más ampliamente utilizados para cada caso:

- *Top Trading Cycle* de David Gale, para el caso en que hay preferencias sólo de un conjunto sobre otro.
- *Algoritmo de Aceptación Diferida*, en el caso en que ambos conjuntos tengan preferencias sobre el otro.

11 Otros algoritmos en networkx

Networkx es una herramienta potentísima y existen muchos otros algoritmos que aquí no se han comentado por ser, en principio, más irrelevantes. Sí se enlistarán los objetivos que cada tipo de algoritmo tiene para, en caso de haber infraestimado su relevancia, poder adjuntar al presente documento una breve explicación de los mismos y, en cualquier caso, que esta breve guía del análisis de redes esté completa en cuanto a lo que se puede-llegar-a-hacer *directamente*, al menos en uno de los softwares comparados, con la intención que en la introducción se especifica.

Así, los tipos de algoritmos restantes y una breve explicación son los siguientes:

- Sparsifiers, *para crear grafos dispersos a partir de otros*.
- Vitality, *se computa la medida de vitalidad²⁸ de un nodo*.
- Cores, *algoritmos para encontrar k-cores²⁹*.
- Covering, *funciones relacionadas con la cobertura de un grafo*.
- Cuts, *funciones para evaluar y evaluar cortes en un grafo*.
- Directed Acyclic Graphs, *algoritmos para grafos dirigidos acíclicos³⁰*.
- Dispersion, *calcula la dispersión entre dos nodos*.
- Non Randomness, *calcula la medida de no-aleatoriedad³¹ de un grafo*.
- Cliques, *funciones en lo referente a encontrar y analizar cliques*.

²⁸La closeness-vitalidad de un nodo v es el cambio en la suma de distancias entre todos los nodos al eliminar dicho nodo v .

²⁹Un k -core de un grafo G es un subgrafo maximal conexo de G cuyos nodos tienen grado al menos k .

³⁰Un grafo dirigido acíclico es un grafo dirigido finito que no contiene ciclos dirigidos.

³¹Las medidas de no-aleatoriedad tratan de medir cuál es la dispersión causada por el azar. En concreto, la utilizada aquí se calcula como $R_G = \sum_{i=1}^k \lambda_i$, la suma de los k (número de comunidades) mayores autovalores de la matriz de adyacencia.

- Components, *funciones para analizar componentes*³² de un grafo.
- Isomorphism, *devuelve si un grafo es (o parece) isomorfo a otro.*
- Dominance, *algoritmos acerca de la dominancia*³³ de los nodos.
- Matching, *funciones para computar y verificar asignaciones en un grafo.*
- Minors, *igual que en cliques, encontrar y analizar menores.*
- Operators, *operaciones en los grafos (composición, unión,...)*
- Planarity, *comprobar si un grafo es planar.*
- Swap, *intercambia aristas en un grafo.*
- Traversal, *algoritmos básicos para la búsqueda en profundidad.*
- Voronoi Cells, *se computan las células de Voronoi*³⁴ de un grafo.

³²Un componente de un grafo es un subgrafo en el que cualesquiera dos nodos están conectados por un camino.

³³Se dice que un nodo d domina a otro n si todo camino dirigido a n debe pasar por d con anterioridad.

³⁴Un diagrama de Voronoi es la partición de un plano en regiones basada en la distancia a los puntos en un subconjunto específico del plano. Esas regiones se llaman células de Voronoi.

Part VI

Problemas en el análisis de redes

12 Enunciado y *solución* de los problemas

12.1 De enumeración

Los problemas de enumeración en grafos son un tipo de problemas de combinatoria en los que el objetivo es contar cuánto grafos son posibles, dirigidos o no dirigidos, en función del número de nodos de los mismos. Pueden ser resueltos analítica o asintóticamente. Los resultados más importantes se trasladan aquí:

- El número de grafos simples no-dirigidos distintos de n nodos es $2^{\frac{n(n-1)}{2}}$.
- El número de grafos simples dirigidos distintos de n nodos es $2^{n(n-1)}$.
- El número C_n de grafos conexos no dirigidos de n nodos sigue la relación de recurrencia $C_n = 2^{\binom{n}{2}} - \frac{1}{n} \sum_{k=1}^{n-1} k \binom{n}{k} 2^{\binom{n-k}{2}} C_k$.
- El número de árboles distintos con n nodos es n^{n-2} , *fórmula de Cayley*.
- El número de árboles tipo caterpillar de n nodos es $2^{n-4} + 2^{\frac{n-4}{2}}$.

Al tratarse de un problema teórico no tiene sentido plantearnos su computabilidad.

12.2 Subgrafos, subgrafos inducidos y menores

Un problema común es el llamado **problema del isomorfismo del subgrafo**: sean dos grafos G y H y debe determinarse si G contiene un grafo isomorfo³⁵ a H .

La razón principal para interesarse por este problema es por ver si las propiedades de los grafos son *hereditarias* para sus subgrafos. Sin embargo, dicho problema es usualmente NP-completo.

Problemas similares son encontrar **subgrafos inducidos** (grafo formado por un subconjunto de los nodos del grafo original y todas las conexiones entre

³⁵En teoría de grafos, un grafo G es isomorfo a otro H si $\exists f$ biyección $f : V(G) \rightarrow V(H)$ tal que $\forall u, v$ adyacentes en G , $f(u), f(v)$ son adyacentes en H .

ellos) de un grafo dado, por ejemplo: encontrar el mayor grafo inducido que no contenga aristas o *independent set problem*.

Otro problema similar refiere a los **menores**. Un menor de un grafo es cualquier grafo obtenido mediante la eliminación de aristas del grafo original o la contracción de los mismos. De nuevo, el objetivo de su estudio es comprobar si las propiedades son heredables. Un resultado que utiliza este tipo de estructura es el Teorema de Wagner: “*Un grafo es planar si no contiene como un menor el grafo completo bipartito $K_{3,3}$ ni el grafo completo K_5 .*”

En el mismo sentido, con el concepto de **subdivisión** o homorfismo de un grafo: un grafo obtenido subdividiendo algunas aristas, se consigue el Teorema de Kuratowski: “*Un grafo es planar si no contiene como subdivisión el grafo completo bipartito $K_{3,3}$ ni el grafo completo K_5 .*”

Un último problema de este tipo es la **conjetura de reconstrucción** de Kelly y Ulam: dado un grafo $G = (V, E)$, se considera el conjunto de subgrafos inducidos resultantes de eliminar exactamente un nodo de G y lo llamamos $D(G)$, *deck* de G . Cada elemento del *deck* es llamado *card*. Dos grafos que sean a su vez elementos del mismo $D(G)$ se dicen que son hipomorfos.

La conjetura de reconstrucción establece: *cualesquiera dos grafos hipomorfos en al menos tres nodos son isomorfos.*

12.3 Coloreado de grafos

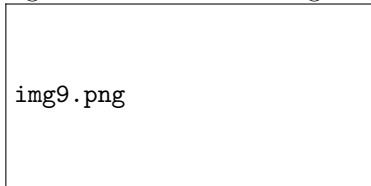
Otro problema clásico es el de coloreado de grafos. El objetivo del problema de este tipo es dilucidar si puede colorearse cierto grafo con ciertas características de tal manera que dos nodos adyacentes no tengan el mismo color, o una restricción similar.

Este problema de coloreado puede trasladarse a realidades más interesantes entendiendo el concepto de *color* con otro tipo de atributos. Un ejemplo sería el problema en que se desea ubicar a ciertos presos en las cárceles, de manera que los colores sean organizaciones criminales, y se pretenda evitar que estén juntos dos presos con el mismo color (de la misma organización).

Un resultado clásico a este respecto es el Teorema de 4-colores: un plano separado por regiones contiguas no requiere de más de cuatro colores para ser coloreado sin que dos regiones adyacentes tengan el mismo color. Dicho teorema se aplica a grafos entendiendo el mapa en conjunto como un planar.

Una conjetura más sofisticada sería la de Hadwiger: si un grafo G no tiene

Figure 1: Puentes de Königsberg



loops³⁶ y no tiene un minor K_t , su número crómico³⁷ es $\chi(G) < t$. Esta conjetura está demostrada para $t \in [1, 6]$.

En `networkx` podemos encontrar una función integrada llamada `greedy_color(G)`, con distintas opciones de estrategia, que colorea un grafo mediante un algoritmo voraz, tratando de utilizar los menos colores posibles. Además, posee también la función `equitable_color(G, num_colors)` donde, dado un Grafo y el número de colores deseado, se trata de colorear el grafo con dicho número de colores de manera que cada color *pinte* aproximadamente el mismo número de nodos.

12.4 Problemas de ruta

Se enumerarán una serie de problemas distintos y su solución, si es el caso, acerca de distintos aspectos que tendrán que ver con los caminos de un nodo a otro del grafo.

12.4.1 Siete puentes de Königsberg

Es el problema clásico de esta rama de la teoría de grafos. El problema fue resuelto negativamente por Euler en 1736 y se enuncia de la siguiente manera: “Dada la división de Königsberg por dos ríos que son cruzados por siete puentes, ¿Existe una ruta que pase por todos los puentes una y solo una vez?”

12.4.2 Problema del camino hamiltoniano

Este problema se basa en determinar si existe un camino hamiltoniano en un grafo dado. Un camino hamiltoniano es aquel que visita cada nodo exactamente una vez. También puede querer encontrarse ciclos hamiltonianos, que sería encontrar un camino infinito que recorra todos los nodos del grafo una y otra vez.

³⁶Conexiones de un nodo con sí mismo.

³⁷Número de colores necesarios para pintar el grafo sin repetir un mismo color en nodos adyacentes.

Algorithm 7 Pettie y Ramachandran para spanning trees.

Sea $r = \log \log \log n$, con n el número de nodos. Se encuentran todos los árboles de decisión óptimos en r nodos.

Se particiona el grafo en componentes con, como mucho, r nodos en cada componente.

Se utiliza el árbol de decisión óptimo para encontrar el spanning tree maximal para cada componente.

Se contrae cada componente conectado por el spanning tree maximal a un único nodo.

Es un problema NP-completo en general, aunque para algunos casos especiales puede ser resuelto en tiempo polinomial.

Hay diferentes algoritmos para buscar la solución, desde la fuerza bruta, el algoritmo enumerativo de Martello, un procedimiento de búsqueda de Frank Rubin que divide las aristas en clases, la programación dinámica de Bellman o un algoritmo basado en el principio de inclusión-exclusión de Andreas Björklund.

En `networkx` se puede tratar de encontrar un camino hamiltoniano a través de `hamiltonian_path(G)`.

12.4.3 Mínimo spanning tree

El spanning tree mínimo se refiere a aquel subconjunto de aristas de un grafo conexo no dirigido que conecta todos los nodos del mismo pero que no tiene ningún ciclo tal que el peso total (o número en caso de un grafo *unweighted*) sea mínimo. Hay diferentes utilidades para este tipo de problemas como puede ser una compañía de telecomunicaciones que desea conectar todas las viviendas entre sí con el menor coste (pesos) posible o, en una visión puramente analítica, como herramienta para la clusterización de la red.

Su solución puede encontrarse a través de distintos algoritmos que, como mínimo, se resuelven en tiempo polinomial. Por ejemplo, el de Bernard Chazelle, basado en comparación. Además, S. Pettie y V. Ramachandran enunciaron un algoritmo *probablemente* óptimo. La descripción básica es la descrita en la figura del Algoritmo 7.

El tiempo computacional es $\mathcal{O}(m)$ para cada paso, excepto en el uso de árboles de decisión, del que no se sabe. Sin embargo, se sabe que es óptimo en el sentido de que ningún algoritmo puede realizarlo mejor que un árbol de decisión óptimo.

En `networkx` están integradas las funciones para encontrar spanning trees mínimos y máximos, así como para generar aristas en un bosque spanning mín-

imo o máximo:

- $minimum_spanning_tree(G)$
- $maximum_spanning_tree(G)$
- $minimum_spanning_edges(G)$
- $maximum_spanning_edges(G)$

En igraph también puede encontrarse, dado un grafo, su spanning tree mínimo.

12.4.4 Problema de inspección de la ruta

El objetivo de este tipo de problemas es encontrar el camino cerrado o *circuito* más corto que visita cada arista del grafo no dirigido. Cuando el grafo tiene un circuito de Euler (un camino cerrado que cubre cada arista una vez), esa es la solución óptima.

De otra manera, el problema de optimización es encontrar el menor número de aristas que se recorren más de una vez. Este problema puede ser resuelto en tiempo polinomial.

La idea de la solución es la misma para grafos dirigidos o no dirigidos y se basa en el concepto de *T-join*. Sea T un subconjunto de nodos del grafo. Un conjunto de aristas J se dice que es *T-join* si la colección de nodos que tienen un número par de aristas de vecinos en J es exactamente el conjunto T .

El problema de la *T-join* es encontrar aquella *T-join* con el menor número posible de aristas.

Para todo subconjunto T , su *T-join* más pequeña (cuando existe) necesariamente consiste en $\frac{1}{2}|T|$ caminos que unen los vértices de T por pares. En una solución óptima, ninguno de estos caminos compartirán aristas, pero sí es posible que compartan nodos.

La mínima *T-join* alcanzable se puede conseguir construyendo un grafo completo en los nodos de T , con aristas que representan los caminos más cortos en el grafo dado como input. Luego se encuentra un match perfecto con peso mínimo en este nuevo grafo.

Al deshacer el cambio de aristas del nuevo grafo a caminos en el input, encontramos el camino más corto que visita cada arista del mismo.

Para este problema, T debe ser escogido como el conjunto de todos los nodos con grado par y el tiempo computacional es $\mathcal{O}(n^3)$ para el proceso completo.

12.4.5 Camino más corto

El problema del camino más corto es uno de los más importantes del análisis de redes. Se define, por simplicidad, para grafos no dirigidos. En grafos dirigidos es un caso análogo, con alguna salvedad.

Sea un camino $P = (v_1, v_2, \dots, v_n) \in V \times V \times \dots \times V$ tal que v_i es adyacente a v_{i+1} , para $1 \leq i < n$. P se dice entonces que tiene longitud $n - 1$.

Sea ahora e_{ij} la arista incidente de v_i a v_j . Dada una función de pesos evaluada en los reales: $f : E \rightarrow \mathbb{R}$, y sea G un grafo simple no dirigido, el camino más corto de v a v' es el camino $P = (v, \dots, v')$ tal que, minimiza la suma $\sum_{i=1}^{n-1} f(e_{i,i+1})$, para cierta n dependiente del camino.

Los problemas, además, pueden ser de varios tipos: encontrar el camino más corto entre un par de nodos (*pair shortest path*), encontrar los caminos más cortos a todos los nodos desde un nodo-fuente dado (*single-source shortest path*) o encontrar los caminos más cortos para cada par de nodos en el grafo (*all pairs shortest paths*).

Los algoritmos más importantes para resolver este problema son los siguientes:

- Dijkstra: single-source para aristas con peso no negativo.
- Bellman-Ford: single-source para cualquier peso de arista.
- Búsqueda A*: para pares de nodos, heurístico.
- Floyd-Warshall: all-pairs.
- Johnson: all-pairs, más rápido si la matriz es sparse.
- Viterbi: all-pairs, para caminos estocásticos donde hay pesos de probabilidad.

Tanto en `igraph` como en `networkx` encontramos varias funciones para encontrar el camino más corto.

En `networkx`, la general, es `shortest_path(G, source=None, target=None, method="dijkstra")`, el método puede ser también Bellman-Ford. Además están integradas funciones que devuelven todos los caminos más cortos, la longitud del más corto (y de todos), la media de la longitud del camino más corto, una función para decir si existe camino, una interfaz avanzada y otros algoritmos más específicos como el de la A*.

En `igraph` se tiene la función `shortest_paths_dijkstra(source=None, target=None)`

12.4.6 Problema del árbol de Steiner

La formulación del problema es la siguiente: dado un grafo no dirigido con pesos no-negativos en las aristas y un subconjunto de nodos (llamados *terminales*), el problema del árbol de Steiner es encontrar aquel árbol que tenga el mínimo peso posible y que contenga todos los nodos del subconjunto, pero pudiendo contener nodos de fuera.

Hay otras formulaciones del problema como el *Problema del árbol de Steiner euclídeo* o el *Problema del árbol de Steiner rectilíneo*.

Se puede hallar una aproximación de la solución para el caso general computando el mínimo spanning tree del subgrafo de la clausura métrica del grafo inducido por los vértices terminales. La clausura métrica de un grafo G es el grafo completo en que cada arista tiene un peso referente a la longitud del camino más corto entre los nodos en G .

Este algoritmo produce un árbol cuyo peso es $2 - \frac{2}{t}$ veces el peso del árbol óptimo de Steiner y se puede encontrar en tiempo polinomial.

Otros algoritmos algo más sofisticados mejoran ese ratio hasta $\ln(4) + \varepsilon \leq 1.39$ a través de la randomización, como es el algoritmo de Byrka et al. (2010).

En este apartado solo encontramos funciones en `networkx`, que son dos: `metric_closure(G)` y `steiner_tree(G, terminal_nodes)` que devuelven, respectivamente, la medida de la clausura métrica del grafo y una aproximación al árbol de Steiner mínimo como se especificó anteriormente.

12.4.7 Problema del vendedor

El problema del vendedor es otro problema clásico de la optimización. El enunciado, informalmente, es el siguiente: “Dada una lista de ciudades y las distancias entre cada par de ciudades, ¿cuál es la posible ruta más corta en que se visita cada ciudad y se vuelve a la ciudad original?”.

El problema es computacionalmente difícil, pero se conocen una gran cantidad de algoritmos heurísticos y exactos debido a su relevancia. Por ese motivo, aquí se plantearán los más conocidos, así como los que tienen un mayor interés teórico.

El grafo en cuestión que plantea el problema es no dirigido y con pesos, donde cada nodo es una ciudad y cada peso es la distancia entre las ciudades

que une la arista correspondiente.

Formalmente, existen dos formulaciones del problema que se suelen utilizar. Una es la de Miller-Tucker-Zemlin:

Sean las ciudades etiquetadas como $1, \dots, n$. Se define:

$$x_{ij} = \begin{cases} 1 & \text{el camino se dirige de la ciudad } i \text{ a } j \\ 0 & \text{otro caso} \end{cases}$$

Para cada $i = 1, \dots, n$, sea u_i una variable dummy y c_{ij} la distancia de la ciudad i a la ciudad j . La formulación para programación entera puede ser:

$$\begin{cases} \min \sum_{i=1}^n \sum_{j \neq i, j=1}^n c_{ij} x_{ij} \\ s.t. \begin{cases} x_{ij} \in \{0, 1\} & i, j = 1, \dots, n; \\ u_i \in \mathbb{Z} & i = 1, \dots, n; \\ \sum_{i=1, i \neq j}^n x_{ij} = 1 & j = 1, \dots, n; \\ \sum_{j=1, i \neq j}^n x_{ij} = 1 & i = 1, \dots, n; \\ u_i - u_j + nx_{ij} \leq n - 1 & 2 \leq i \neq j \leq n; \\ 0 \leq u_i \leq n - 1 & 2 \leq i \leq n. \end{cases} \end{cases}$$

La **computación** de la **solución** puede conseguirse a través de: algoritmos exactos, aproximaciones heurísticas o casos especiales.

En el caso de los algoritmos **exactos**, probar todas las permutaciones (*fuerza bruta*), conlleva un tiempo computacional de $\mathcal{O}(n!)$. Mientras que un algoritmo basado en programación dinámica de Held-Karp alcanza $\mathcal{O}(n^2 2^n)$.

En cuanto a los algoritmos **aproximativos**, hay diferentes tipos de los mismos y la solución es razonablemente buena para cada caso.

Heurística constructiva

En este tipo de algoritmos se encuentra el Nearest Neighbour (*vecino más cercano*). Se trata de un algoritmo voraz o *greedy*. Este algoritmo escoge como siguiente nodo en el camino a aquel otro a menor distancia que no haya sido visitado. Como es lógico, la solución dependerá del punto de inicio del camino. El tiempo computacional es $\mathcal{O}(\log |V|)$.

Algoritmo de Christofides

Se parte en la idea de encontrar ciclos eulerianos y hamiltonianos. Un ciclo euleriano es aquel que pasa por cada arista una vez y sólo una vez y un ciclo hamiltoniano es aquel que visita todos los vértices del grafo sólo una vez, a

Algorithm 8 Algoritmo de Christofides

Se obtiene el spanning tree mínimo T de G .

Sea O el conjunto de vértices de grado impar en T , hallar un apareamiento perfecto de M de mínimo peso en el grafo completo sobre los vértices de O .

Se combinan las aristas de M y T para crear un multigrafo H .

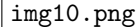
Se obtiene un ciclo euleriano en H .

Se obtiene un ciclo hamiltoniano a partir del anterior, descartando los nodos visitados.

excepción del nodo inicial/final. Puede verse la descripción del algoritmo en el algoritmo 8. Puede demostrarse que, como máximo, la solución obtenida por el algoritmo es 1.5 veces la óptima a través del uso de la desigualdad triangular para los pesos de ciertos subconjuntos creados a través de árboles recubridores o *spanning tree*.

Intercambio de par

Es un caso particular del algoritmo k-opt. Se basa en eliminar dos aristas en el camino para reconectar los fragmentos intercambiando la unión de los nodos, como en la siguiente imagen:



img10.png

La generalización k-opt es la misma idea pero eliminando y reconstruyendo k conexiones. Una generalización posterior del k-opt sería el v-opt. En este algoritmo, el tamaño del set de aristas eliminadas no es una constante, sino que crece con el proceso.

El tiempo computacional para el caso habitual, el 2-opt, varía conforme comencemos con una solución inicial a partir del algoritmo voraz, $\mathcal{O}(n)$ o de una solución aleatoria, $\mathcal{O}(n \log(n))$.

Colonia de hormigas

El algoritmo de la colonia de hormigas es un algoritmo probabilístico que se basa en el comportamiento de las hormigas en el mundo real. Se crean hormigas artificiales a las que se les enfrenta en un escenario en el que tratan de encontrar el camino más rápido entre un punto y otro.

La idea es la siguiente: supongamos un camino que, en un cruce se convierte en dos y ambas opciones son *no exploradas* por ninguna hormiga, uno sensiblemente más larga que la otra.

Se somete a un grupo suficientemente grande de hormigas al escenario y, éstas, **aleatoriamente**, escogen uno de los dos caminos.

La clave del algoritmo radica en que las hormigas segregan feromonas - suponemos uniformemente- en su trayecto. No es que las hormigas *aprendan* individualmente qué opción es mejor, sino que acaban por seguir aquella con mayor concentración de feromonas.

La concentración de feromonas será superior en el caso del camino más corto posible porque las hormigas *rápidas dan más viajes* entre el hormiguero (nodo inicial) y la comida (nodo final). Desde luego, para esto es necesario que la concentración de hormigas esté distribuida uniformemente en cada opción.

Formalmente, el comportamiento de una hormiga artificial es el descrito a continuación. Para cada iteración del algoritmo, cada hormiga k se moverá de un nodo x a otro y . Para ello computa un set $A_k(x)$ de nodos alcanzables, desplazándose al nodo y con probabilidad p_{xy}^k . Dicha probabilidad depende de la combinación de dos valores: la atracción η_{xy} computada por algún heurístico que establece *a priori* la adecuación de ese desplazamiento y el nivel del sendero τ_{xy} del movimiento, que indica cómo de eficiente fue tomar ese camino en el pasado, aceptando el nivel de feromona como su proxy.

En general, la probabilidad será:

$$p_{xy}^k = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum_{z \in A_k(x)} (\tau_{xy}^\alpha)(\eta_{xy}^\beta)}$$

El parámetro $\alpha \geq 0$, $\alpha \in \mathbb{R}$ es un control de influencia sobre la cantidad de feromonas. Análogamente, $\beta \geq 1$ lo es para η_{xy} , el conocimiento *a priori*. Dicho conocimiento *a priori* suele establecerse como $\eta_{xy} = \frac{1}{d_{xy}}$, con d_{xy} los pesos.

Cuanto todas las hormigas completan una iteración, los rastros de feromonas son actualizados por:

$$\tau_{xy} \leftarrow (1 - \rho)\tau_{xy} + \sum_k \Delta\tau_{xy}^k$$

donde ρ es el coeficiente de evaporación de feromonas y $\Delta\tau_{xy}^k$ es la cantidad de feromonas depositadas por la k -ésima hormiga. Viene dado típicamente por:

$$\Delta\tau_{xy}^k = \begin{cases} \frac{Q}{L_k} & \text{si } k \text{ utiliza el camino } xy \\ 0 & \text{otro caso} \end{cases}$$

donde L_k es el coste de la ruta de la k -ésima hormiga y Q es una constante.

12.5 Problemas de visibilidad

El clásico problema de visibilidad es el problema de la galería de arte *o del museo*. El objetivo es encontrar el mínimo número de guardias necesarios de tal manera que, juntos, puedan observar toda la galería a la vez. La solución, para el plano, viene dada por Chvátal y Fisk, que demuestran que $\frac{n}{3}$ son el número máximo de guardas necesarios.

Computacionalmente es un problema NP-Hard en casi todas sus versiones. Existen algoritmos aproximativos y exactos, como el de Couto, de Rezende y de Souza (2011), que se divide en dos fases: una primera fase de preproceso, donde se discretiza la región convirtiéndola en un planar y una fase de solución iterativa, cuyo problema de optimización es:

$$\begin{cases} z = \min \sum_{j \in V} x_j \\ \text{s.t.} \begin{cases} \sum_{j \in V} a_{ij} x_j \geq 1 & \forall p_i \in D(P) \\ x_j \in \{0, 1\} & \forall j \in V \end{cases} \end{cases}$$

donde $D(P)$ es la partición del planar, x_j es la variable binaria que vale 1 si se localiza un guardia en un nodo j y 0 en caso contrario y, dado un punto p_i y un nodo j , a_{ij} es una variable binaria que vale 1 si $p_i \in Vis(j)$ ³⁸, 0 en caso contrario.

12.6 Problemas de descomposición

Este tipo de problemas tienen como objetivo reducir la complejidad del grafo para obtener una forma reducida, más simple, del mismo. Podemos verlo como el equivalente a la reducción dimensional del análisis multivariante (análisis factorial, componentes principales, etc). Con estos grafos simplificados podemos mejorar la comprensión de los mismos, además de comprobar las estructuras que subyacen en ellos. De esto se encarga la factorización de grafos.

Además veremos uno de los problemas más importantes de este tipo: la arboricidad.

Para esta tarea existe en networkx una función llamada `chain_decomposition(G)` que devuelve la descomposición en cadenas de dicho grafo.

³⁸Región de visibilidad

Figure 2: 1-factorización de Desargues

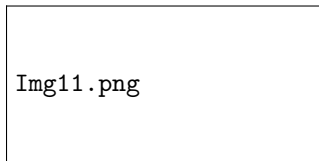
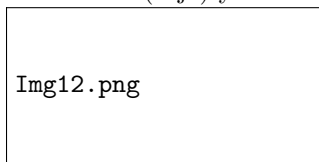


Figure 3: Un 1-factor (rojo) y un 2-factor (azul)



12.6.1 Factorización de un grafo

Definimos un factor de un grafo G como un subgrafo que tiene el mismo conjunto de nodos que G . Desde aquí podemos definir un k -factor como el subgrafo spanning k -regular³⁹, con k el número de vecinos de cada nodo.

Una k -factorización particiona las aristas de un grafo en k -factores disjuntos. Podemos ver dos ejemplos para una comprensión más clara del significado de los factores en las figuras 2 y 3.

12.6.2 Arboricidad

La arboricidad de un grafo no-dirigido es el número de bosques en los que sus aristas pueden ser particionadas. Equivalentemente, es el mínimo número de spanning forests que se necesitan para cubrir todas las aristas del grafo.

El teorema de Nash-Williams (1964) establece:

Un grafo G es t -arboresco⁴⁰ si y sólo si, $\forall U \subset V(G)$ el grafo inducido $G[U]$ tiene tamaño $|G[U]| \leq t(|U| - 1)$.

La arboricidad de un grafo puede servir como medida de densidad: grafos con muchas aristas tienen alta arboricidad, por lo que deberán ser grafos densos.

Para hallar la arboricidad podemos utilizar algoritmos destinados a la partición de matroides, ya que la arboricidad puede expresarse como un caso especial de esta.

Así, podemos considerar el problema de un matroide donde se desea expresar

³⁹Un grafo es regular si todos los nodos tienen el mismo número de vecinos.

⁴⁰Un grafo se dice t -arboresco si puede ser particionado en t aristas disjuntas.

un conjunto de elementos del mismo como una unión de un número pequeño de conjuntos independientes. Existe, en este sentido, un algoritmo computable en tiempo polinomial de Gabow y Westerman (1992).

12.7 Flow Network

Una red de flujo, también llamada red de transporte, es un grafo dirigido donde cada arista tiene una capacidad y dada arista recibe cierto flujo que no puede exceder la capacidad. Dicho flujo comienza en la fuente (nodo inicial) y tiene como destino el sumidero (nodo final). Los problemas usuales asociados a este tipo de grafos son los siguientes:

12.7.1 Flujo máximo

Es el problema clásico en las redes de transporte, además del más común. El objetivo es encontrar aquel camino que permita maximizar el flujo transportado desde la fuente al sumidero. Para conseguir este objetivo existen algunos algoritmos como son:

- Relabel-to-front ($\mathcal{O}(n^2m)$)⁴¹

Este algoritmo se basa en dos operaciones: *Push*, cuando un nodo tiene exceso de flujo y existe un nodo adyacente con un menor valor *height*⁴² de éste, se traspasa flujo del primero al segundo. Y *relabel*, si un nodo tiene exceso de flujo pero no tiene ningún nodo adyacente con un menor valor, se aumenta el valor *height* del nodo de tal manera que pueda realizar la función push.

- Edmons-Karp ($\mathcal{O}(m^2n)$)

Este algoritmo se basa en el método de Ford-Fulkerson, cuya idea es, brevemente: si existe un camino desde la fuente al sumidero con capacidad disponible en todas las aristas del camino, se envía flujo a través de dicho camino. Luego encontramos otro camino y así. Edmons-Karp lo que hacen es concretar cómo encontrar estos caminos con capacidad disponible: mediante breadth-first search (*búsqueda en anchura*).

- MPM ($\mathcal{O}(n^3)$)

⁴¹El número de nodos es n y de aristas m .

⁴²La función *height* se denota $l : V \rightarrow \mathbb{N}$ y debe cumplir: i) $l(u) \leq l(v) + 1 \ \forall u, v \in E_f$, $l(s) = |V|$, $l(t) = 0$.

Este algoritmo es implementado por fases, durante cada una se considera la red por capas L y se encuentran pontencial-interior y pontencial-exterior, p_{in} ⁴³ y p_{out} ⁴⁴. Con ellos se define el potencial $p(v) = \min(p_{in}(v), p_{out}(v))$. Se denomina referencia, r , al nodo con menor pontencial. Entonces se incrementa el flujo en $p(r)$, pues puede ser absorbido por las demás aristas. En cada fase se repite el proceso de búsqueda de r mientras se eliminan las aristas totalmente saturadas.

- James Orlin ($\mathcal{O}(mn)$)

El algoritmo resuelve a través de mejora iterativa. El input para cada fase de mejora es un flujo x y un vector $r = r(x)$ de capacidades residuales y un corte⁴⁵ (S, T) , nodo-fuente. Se define $\Delta = r(S, T)$, cota máxima del flujo residual de s a t . El output de esta fase es un flujo x' , un vector $r' = r(x')$ de capacidades residuales y un corte (S', T') .

En este aspecto, podemos encontrar en networkx distintas funciones para analizar los flujos. Estas son:

- `maximum_flow(flowG, _s, _t)` - `_s` y `_t` refieren a fuente y sumidero.
- `maximum_flow_values(flowG, _s, _t)`
- `minimum_cut(flowG, _s, _t)`
- `minimum_cut_value(flowG, _s, _t)`
- `edmons_karp(G, s, t)`
- `shortest_aumenting_path(G, s, t)`
- `preflow_push(G, s, t)`
- `dinitz(G, s, t)`
- `boykov_kolmogorov(G, s, t)`
- `gomory_hu_tree(G)`
- `build_residual_network(G, capacity)`

⁴³ $p_{in}(v) = \sum_{u,v \in L} [c(u, v) - f(u, v)]$

⁴⁴ $p_{in}(v) = \sum_{u,v \in L} [c(v, u) - f(v, u)]$

⁴⁵Un corte es una partición de los nodos de un grafo en dos subconjuntos disjuntos. Un corte s - t requiere que fuente y sumidero estén en distintos subconjuntos.

- *network_simplex*(*G*, *demand*, *capacity*, *weight*)
- *min_cost_flow_cost*(*G*, *demand*, *capacity*, *weight*)
- *min_cost_flow*(*G*, *demand*, *capacity*, *weight*)
- *cost_flow*(*G*, *flowDict*, *weight*)
- *max_flow_min_cost*(*G*, *s*, *t*, *capacity*, *weight*)

En *igraph* encontramos otras como *maxflow*(*source*, *target*) y *gomory_hu_tree*(*capacity*, *flow*="flow").

12.7.2 Otros

Otros problemas posibles en este campo serían el problema **multi-commodity**, donde se tienen varias fuentes y sumideros, así como varias materias (*commodities*) que deben ir de determinada fuente a un sumidero dado o el problema del **mínimo coste** donde, cada arista \overrightarrow{uv} tiene un coste asociado $k(u, v)$ y el objetivo final es minimizar el coste total entre fuente y sumidero. El campo es amplio y existen numerosos problemas adicionales.

12.8 Análisis del Camino Crítico y Evaluación de Programa (PERT)

12.8.1 Análisis del camino crítico

Por una parte, el análisis del camino crítico es un algoritmo para programar un conjunto de actividades. Un camino crítico se determina identificando el tramo más largo de caminos dependientes y midiendo el tiempo requerido para completarlas desde el inicio al final.

12.8.2 Evaluación de programa (PERT)

PERT, *program evaluation and review technique*, es un método para analizar las tareas involucradas en la compleción de un programa. Se especifica el tiempo de cada tarea y se identifica el tiempo mínimo para completar el conjunto del proyecto.

Con el fin de definir la precedencia entre actividades y crear una malla PERT óptima deben seguirse tres principios:

- **Principio de designación sucesiva:** los nodos son nombrados por números naturales tal que no se les asigna un número hasta que no son nombrados todos aquellos nodos predecesores, en el sentido de nodos que tienen aristas dirigidas a ellos.
- **Principio de unicidad del estado inicial y final:** sólo puede existir un nodo inicial y un nodo final.
- **Principio de designación unívoca:** no pueden existir dos aristas que compartan nodos de origen y destino.

12.9 Otros problemas

12.9.1 Problema de localización

El análisis de localización se ocupa del emplazamiento óptimo de instalaciones con el fin de minimizar los costes de transporte considerando otros factores como evitar materiales peligrosos cerca de residencias y las instalaciones de competidores.

Puede ser resuelto de manera exacta por el *slab dividing* de Hwang en un tiempo computacional elevado, mejorable mínimamente por una aproximación via core-sets de Badoiu que no se separa más que por un factor $1 + \varepsilon$ y, por su simplicidad computacional, el farthest-point clustering, utilizado habitualmente.

12.9.2 Problema de transporte

Partiendo de un concepto similar al anterior, un problema de transporte trata de encontrar la manera óptima de transportar y alojar recursos.

La formulación abstracta del problema se debe a Monge y Kantorovich y es la siguiente.

Sean X y Y dos espacios métricos separables⁴⁶ tal que cualquier medida de probabilidad en X es una medida de Radon⁴⁷. Sea $c : X \times Y \rightarrow [0, \infty]$ una

⁴⁶Un espacio métrico (X, d) con d distancia asociada, es separable si incluye un subconjunto denso numerable. Un conjunto $A \subset X$ es denso en X si $\bar{A} = X$.

⁴⁷Una medida se dice de Radon si es interior-regular:

$$\forall \text{abierto } U, m(U) = \sup_{K \text{ compacto } \subset U} \{m(K)\}$$

exterior-regular:

$$\forall \text{boreliano } B, m(B) = \inf_{B \text{ abierto } \subset U} \{m(U)\}$$

función Borel-medible⁴⁸. Dada una medida de probabilidad μ en X y ν en Y , el problema puede plantearse como encontrar $T : X \rightarrow Y$, función de transporte, que alcanza el mínimo:

$$\inf \left\{ \int_X c(x, T(x)) d\mu(x) \mid T_*(\mu) = \nu \right\}$$

donde $T_*(\mu)$ denota la medida de la imagen⁴⁹ (*pushforward*) de μ .

La solución para la recta real es relativamente sencilla, mientras que en mayores dimensiones debe recurrirse a ciertas propiedades de los espacios de Hilbert separables.

13 Propagación de contenido

La propagación de contenido en una red puede producirse de dos formas distintas: conservativa o no conservativa. La primera se da cuando la cantidad de información en la red es constante según va recorriéndola, como podría ser cierta cantidad de agua en un sistema de tuberías. La no-conservativa, sin embargo, puede aumentar o decrecer en tamaño. Se puede modelizar así, por ejemplo, las enfermedades infecciosas.

13.1 El modelo SIR

El modelo SIR, creado en 1927 por W. O. Kermack y A. G. McKendrick considera una población dada donde se transmite una enfermedad y se definen tres variables en el tiempo:

- $S(t)$, susceptibles: representa el número de individuos aún no infectados con la enfermedad en tiempo t .

y localmente finita:

$$\forall x \in X, \exists V, \text{ bola abierta centrada en } x : m(V) < \infty$$

⁴⁸Una función $f : (X, \Sigma) \rightarrow (Y, T)$ es Borel-medible si $\forall B \subset Y, f^{-1}(B)$ es medible y $(X, \Sigma), (Y, T)$ son espacios de Borel.

⁴⁹Dados dos espacios medibles (X_1, Σ_1) y (X_2, Σ_2) , una aplicación medible $f : X_1 \rightarrow X_2$ y una medida $\mu : \Sigma_1 \rightarrow [0, \infty]$, la medida de la imagen $f_{\#}\mu$, de μ viene dada por:

$$f_{\#}\mu(B) = \mu(f^{-1}(B)) \quad \forall B \in \Sigma_2$$

- $I(t)$, infectados: representa el número de individuos infectados con la enfermedad y que pueden infectar a aquellos susceptibles.
- $R(t)$, recuperados: representa el número de individuos por el que pasó la enfermedad. No pueden transmitirla ni contagiarse de nuevo.

El flujo en el modelo puede considerarse: $\mathcal{S} \rightarrow \mathcal{I} \rightarrow \mathcal{R}$. Dada una población estática⁵⁰: $N = S(t) + I(t) + R(t)$. De donde se pueden derivar, bajo ciertos supuestos, las siguientes ecuaciones diferenciales:

- $\frac{dS}{dt} = -\beta SI$
- $\frac{dI}{dt} = \beta SI - \gamma I$
- $\frac{dR}{dt} = \gamma I$

Siendo β la probabilidad *frecuentista* de contraer la enfermedad, y γ , la tasa de recuperación. Se conocen, gracias a Miller, las soluciones analíticas del problema:

- $S(t) = S(0)e^{-\xi(t)}$
- $I(t) = N - S(t) - R(t)$
- $R(t) = R(0) + N \frac{\gamma}{\beta} \xi(t)$
- $\xi(t) = \frac{\beta \int_0^t I(t^*) dt^*}{N}$

Podemos interpretar $\xi(t)$ como la esperanza del número de transmisiones que un individuo recibe hasta el tiempo t .

Este modelo SIR se encuentra implementado en igrph, pero no para Python, sino únicamente para R.

13.2 La ecuación maestra

Una ecuación maestra puede expresar el comportamiento de una red creciente no direccional donde, en cada instante $t_0 + i$, $i \in \mathbb{N}$ se añade un nuevo nodo. La red inicial suponemos que está formada por dos nodos y dos enlaces entre ellos en $t_0 = 2$.

⁵⁰Existen además, modelos SIS donde se pueda recaer de la enfermedad y otros que incluyen nacimientos y defunciones naturales.

La ecuación maestra para esta red es:

$$p(k, s, t + 1) = \frac{1}{t}p(k - 1, s, t) + (1 - \frac{1}{t})p(k, s, t)$$

donde $p(k, s, t)$ es la probabilidad de tener un nodo s tenga grado k en $t + 1$ y s es el instante en que dicho nodo se añadió a la red. Nótese que sólo hay dos maneras de que s tenga k enlaces en $t + 1$:

1. El nodo s tiene grado $k - 1$ en el tiempo t y se le añade el nuevo enlace creado en $t + 1$ con probabilidad $\frac{1}{t}$.
2. El nodo ya tiene grado k en t y no se enlaza con el nuevo nodo.

Simplificando el modelo, la distribución del grado queda $P(k) = 2^{-k}$.

Basándonos en una red creciente, podemos desarrollar un modelo epidémico siguiendo una simple regla: cada vez que un nodo se añade y tras escoger un nodo ya existente a enlazarse, se toma la decisión de si el nuevo nodo está infectado o no. El modelo seguirá la ecuación maestra:

$$p_r(k, s, t + 1) = \frac{r_t}{t}p(k - 1, s, t) + (1 - \frac{1}{t})p_r(k, s, t)$$

donde r_t representa la decisión de infectar ($r_t = 1$), o no ($r_t = 0$). La solución que se obtiene al resolver la ecuación es la siguiente: $\tilde{P}_r(k) = (\frac{r}{2})^k$.

Part VII

Análisis dinámico de redes

14 Introducción al análisis dinámico

El análisis dinámico de redes (DNA) incorpora al análisis de redes tradicional una variable temporal al grafo, incorporando, subsecuentemente, incertidumbre. Además hay algunas propiedades interesantes que pueden explotarse. Por ejemplo, al contrario que un modelo estático, un modelo dinámico tiene la habilidad de aprender: las propiedades pueden cambiar en el tiempo y los nodos adaptarse.

Formalmente hay tres diferencias que deben entenderse entre las redes sociales dinámicas y las estáticas. Primero, el DNA utiliza las **meta-redes**, que comentaremos a continuación. Segundo, para explorar cómo las redes evolucionan y se adaptan, así como el impacto de intervenciones en las redes deben utilizarse **simulaciones** que, generalmente, se basan en modelización basada en agentes. Por último, los links en la red **no son binarios**, sino que, en muchos casos, representan la probabilidad de que haya un link, con una noción similar a la de la de los átomos en la física cuántica.

Una **meta-red** o *meta-network*, es una red multi-modo⁵¹, multi-link⁵² y multi-nivel⁵³.

14.1 Sistema dinámico de grafos

Un sistema dinámico de grafos (GDS) puede definirse informalmente como un conjunto *generalmente finito* de grafos con un espacio de estados *generalmente finito*⁵⁴.

Formalmente, un GDS está formado por los siguientes elementos.

- Un grafo finito Y con un conjunto de nodos $V(Y) = \{1, 2, \dots, n\}$, pudiendo ser dirigido o no.

⁵¹Una red es multi-modo si existen diferentes tipos de nodos (e.g., personas y coches).

⁵²Una red multi-link es aquella con distintos tipos de relaciones (e.g., amistad y enemistad).

⁵³Una red multi-nivel es aquella donde algunos nodos son miembros de otros nodos (e.g., una red compuesta por personas y organizaciones donde las personas pueden ser miembro de organizaciones).

⁵⁴En teoría se puede definir ambos espacios como infinitos, pero en la práctica se asume implícitamente finitud.

- Un estado x_v para cada nodo v de Y tomado de un conjunto finito K . El estado del sistema es una n -tupla $x = (x_1, x_2, \dots, x_n)$, y $x[v]$ es la tupla consistente en los estados asociados a los nodos vecinos de v en Y , en algún orden prefijado.
- Una función nodal $f_v \forall v \in V$ que transforma el estado x_v de v en el tiempo, de t a $t + 1$.

$$f_v : X[v] \times T \rightarrow X[v]$$

- Un esquema de actualización especificando el mecanismo por el cual la aplicación al estado de cada nodo individual es llevada a cabo con el fin de inducir un sistema dinámico discreto con una aplicación $F : K^n \rightarrow K^n$.

El espacio de fases asociado al sistema dinámico con aplicación F , es el grafo finito dirigido con un set de nodos K^n y aristas dirigidas $(x, F(x))$.

Este campo replica el estudio del análisis de redes estático habitual y, aunque en un estado relativamente emergente, existe una literatura lo suficientemente amplia. Una referencia de base podría ser el libro de K. M. Carley [61].

Para el análisis dinámico de redes se recomienda usualmente la librería para Python *Dynet*⁵⁵.

15 Predicción de links

15.1 Introducción

La predicción de links (*o relaciones*) es un problema central en el análisis de redes, sobre todo cuando se tratan de redes sociales, aunque no está propiamente dentro del análisis dinámico. La pregunta *informal* que nos cuestionamos es sencilla: ¿Qué relaciones van a crearse en el futuro?

En este sentido, lo que realmente queremos descubrir es hasta qué punto la evolución de una red social puede ser modelada utilizando características intrínsecas a su topología.

Existen múltiples algoritmos para predecir estas relaciones y muchos de ellos vienen implementados en Neo4j y networkx. Antes de su exposición, se definirá el problema de una manera más formal.

⁵⁵Documentación disponible en GitHub: <https://github.com/GiulioRossetti/dynetx>

Figure 4: Predicción de link

img13.png

15.2 Definición del problema

Sea $G = (V, E)$ un grafo no-dirigido, sin pesos. Cada arista $e = \overrightarrow{uv} \in E$ representa la interacción entre el nodo u y v , que se da en un tiempo dado $t(e)$. Dados dos tiempos t, t' , tal que $t' > t$, sea $G(t, t')$ el subgrafo de G consistente en todas las aristas creadas entre t y t' . Sea t_0, t'_0, t_1 y t'_1 cuatro tiempos $t_0 < t'_0 \leq t_1 < t'_1$. El problema de predicción será entonces:

Dada una red $G(t_0, t'_0)$, se devolverá una lista de aristas no presente en $G(t_0, t'_0)$ que se predice que estarán en $G(t_1, t'_1)$.

Para generar la lista, se usan algoritmos heurísticos para asignar una matriz de similitud S cuya entrada $s_{xy} \in \mathbb{R}$ es el score entre los nodos x e y . Todos los links no existentes se ordenan decrecientemente: las relaciones en el top son las más probables a aparecer.

15.3 Medidas de similitud

Hay distintas medidas de similitud con las que conseguir los scores que predicen las conexiones en el grafo y algunas ya se han visto en otros contextos.

15.3.1 Distancia en el grafo

Es la medida directa para comprobar cómo de *cercanos* son dos nodos en un grafo. Se define simplemente como la longitud **negativa**⁵⁶ del camino más corto entre los nodos.

La exploración del grafo en busca de caminos más cortos entre x e y se recomienda de la siguiente manera: se inicializa $S = \{x\}$, $D = \{y\}$. En cada paso se incluyen en el set los vecinos, $\Gamma(x)$, de los nodos en S o en D ⁵⁷, i.e., $S = S \cup \{v | \overrightarrow{uv} \in E \wedge u \in S\}$, así como $D = D \cup \{u | \overrightarrow{uv} \in E \wedge v \in D\}$. El algoritmo se para cuando $S \cap D = \emptyset$. El número de iteraciones será el camino más corto.

⁵⁶Se utiliza un cambio de signo con el fin de que los scores más altos sean aquellos con mayor cercanía (o menor distancia)

⁵⁷Se escogerá expandir aquel set con menor tamaño, por cuestiones de eficiencia.

$$s_{xy} = |sp(x, y)|$$

siendo $sp(x, y)$ el camino más corto entre ambos nodos.

15.3.2 Vecinos comunes

El predictor de los vecinos comunes se basa en la idea de que dos desconocidos que comparten un amigo pueden ser presentados por éste, teniendo el efecto de *cerrar un triángulo*, como en otras medidas ya vistas. Así, el score se define:

$$s_{xy} = \#\{\Gamma(x) \cap \Gamma(y)\}$$

15.3.3 Coeficiente de Jaccard

El coeficiente de Jaccard mide la probabilidad de que tanto x como y tengan un atributo f , para un atributo dado f de x o y .

En este contexto, la característica f son los vecinos, por lo que el score se define:

$$s_{xy} = \frac{\#\{\Gamma(x) \cap \Gamma(y)\}}{\#\{\Gamma(x) \cup \Gamma(y)\}}$$

o, lo que es lo mismo, el número de vecinos compartidos, entre el número de vecinos totales de los nodos.

15.3.4 Adamic-Adar

Esta medida refina el concepto anterior, no simplemente contando los atributos f , sino ponderando aquellos atributos raros con mayor peso.

La noción que subyace detrás, en un contexto social, es que la similitud es más probable si se comparten características raras. Por ejemplo, en una oficina neoyorkina, tenderán a una mayor similitud (*y, por tanto, más probabilidad de relación*) dos trabajadores ilicitanos que dos estadounidenses, con el resto de características iguales. Lo mismo ocurre con los vecinos *poco comunes*.

$$s_{xy} = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log(\#\{\Gamma(z)\})}$$

15.3.5 Conexión preferencial

Un concepto claro en las redes sociales es que, aquel más conocido, se vuelve aún más conocido. Detrás de esta noción subyace la distribución de Pareto, que suele utilizarse para explicar efectos de la generación de riqueza simplificada en la manida “*el rico cada vez se hace más rico*”⁵⁸.

Para estimar dicho efecto, se multiplican los vecinos que tiene cada nodo consiguiendo un efecto de escala. Esta definición de similitud tiene el menor gasto computacional por su sencillez pero, como podemos entender, su adecuación teórica es limitada.

$$s_{xy} = \#\{\Gamma(x)\}\#\{\Gamma(y)\}$$

15.3.6 Conteo del camino alisado exponencialmente de Katz

Este heurístico define una medida que directamente suma colecciones de caminos, alisados exponencialmente por la longitud, de tal manera que tengan mayor peso los caminos más cortos, pues se consideran más importantes en términos de similitud.

$$s_{xy} = \sum_{l=1}^{\infty} \beta^l \#\{Path_{xy}^l\}$$

siendo $\{Path_{xy}^l\}$ el conjunto de caminos de x a y con longitud l . El parámetro de alisado exponencial $\beta \in (0, 1]$ da mayor peso a los caminos más cortos cuanto menor es. Un $\beta = 1$ pondera todos los caminos por igual, independientemente de su longitud.

15.3.7 Hitting time

Supongamos un paseo aleatorio que comienza en el nodo x e, iterativamente, se dirige a un vecino de x , y de ahí a otro vecino y así sucesivamente. Definimos el *hitting time* de x a y como la esperanza del número de pasos que debe dar el camino para alcanzar y por primera vez

⁵⁸Aunque no en este sentido, existe un coeficiente denominado de *rich club* definido por J. McAuley y L. da Fontoura que se define, para cada grado k , como el ratio del número de aristas existentes en nodos con grado mayor o igual que k entre todas las aristas potenciales entre dichos nodos. Concretamente:

$$\phi(k) = \frac{2E_k}{N_k(N_k - 1)}$$

$$h_{xy} = \mathbb{E} [\#\{\overrightarrow{xv_1}, \overrightarrow{v_1v_2}, \dots, \overrightarrow{v_{i-1}y}\}]$$

siendo $(x, v_1, v_2, \dots, v_{i-1}, y)$ un camino aleatorio con $x \neq v_k \neq y \forall k$.

$$s_{xy} = -h_{xy}$$

Un problema que puede darse con esta medida es que h_{xy} puede volverse pequeño si la π_y , probabilidad en el límite de que la cadena se encuentre en el nodo y , independientemente de su relación con x . Con el fin de eliminar este efecto, se pondera h_{xy} que, cuanto menor mayor similitud, por la probabilidad límite⁵⁹.

$$s_{xy} = -h_{xy}\pi_y$$

15.3.8 Rooted Pagerank

Una dificultad que encontramos con el método anterior es que puede variar sensiblemente en función de elementos del grafo muy alejados de ambos nodos x e y , incluso cuando estos son cercanos.

Una forma de evitar este efecto no deseado es resetear el camino aleatorio con cierta recurrencia. En concreto, en cada iteración, la probabilidad de reseteo es $\alpha \in [0, 1) \subset \mathbb{R}$, con el caso $\alpha = 0$ el método *hitting time* anterior.

$$s_{xy} = -h_{xy}\pi_y$$

Con un camino aleatorio tal que:

$$p_{uv} = \begin{cases} \alpha & v = x \\ 1 - \alpha & v \in \Gamma(u) \end{cases}$$

donde p_{uv} es la probabilidad de transición del nodo u a v .

15.3.9 Friends-measure

Cuando se toman dos nodos en una red social, podemos asumir que, a mayor número de conexiones tienen sus vecinos entre sí, mayor probabilidad hay de que dichos nodos estén conectados. Por ello puede definirse:

⁵⁹La probabilidad límite puede calcularse analíticamente considerando el camino una cadena de Markov o estimarse numéricamente.

$$s_{xy} = \sum_{u \in \Gamma(x)} \sum_{v \in \Gamma(y)} \delta(u, v)$$

teniendo la función $\delta(x, y)$ definida como:

$$\delta(x, y) = \begin{cases} 1 & \text{si } x = y \text{ ó } \overrightarrow{xy} \in E \text{ ó } \overrightarrow{yx} \in E \\ 0 & \text{otro caso} \end{cases}$$

15.3.10 Resource Allocation

La similitud a partir de la resource allocation se basa en el inverso de la distancia entre x e y .

$$s_{xy} = \sum_{c \in \Gamma(x, y)} \frac{1}{d(c)}$$

15.3.11 Common Neighbors

La similitud basada en los vecinos comunes es directa. Simplemente es el número de vecinos compartidos:

$$s_{xy} = \#\Gamma(x, y)$$

15.3.12 Inter-cluster common neighbors

Esta similitud parte de la idea de que dos nodos tienen diferentes roles según formen parte o no del mismo cluster, por lo que esta medida se basa tanto en las relaciones de los nodos en el interior del cluster, como de las relaciones de estos con nodos del exterior.

Se parte de la probabilidad bayesiana a posteriori de que x e y pertenezcan al mismo cluster C_α , dados sus sus vecinos comunes $\Lambda_{x, y} = \Gamma(x) \cap \Gamma(y)$ el set de vecinos comunes de los nodos x e y :

$$P(x^{C_\alpha}, y^{C_\alpha} | \Lambda_{x, y}) = \frac{P(\Lambda_{x, y} | x^{C_\alpha}, y^{C_\alpha}) P(x^{C_\alpha}, y^{C_\alpha})}{P(\Lambda_{x, y})}$$

Análogamente, se puede calcular la probabilidad a posteriori de que los nodos pertenezcan a diferentes clusters C_α y C_β .

$$P(x^{C_\alpha}, y^{C_\beta} | \Lambda_{x, y}) = \frac{P(\Lambda_{x, y} | x^{C_\alpha}, y^{C_\beta}) P(x^{C_\alpha}, y^{C_\beta})}{P(\Lambda_{x, y})}$$

A partir de estas probabilidades se deriva la medida de similitud:

$$s_{x,y} = \frac{P(\Lambda_{x,y} | x^{C_\alpha}, y^{C_\alpha}) P(x^{C_\alpha}, y^{C_\alpha})}{P(\Lambda_{x,y} | x^{C_\alpha}, y^{C_\beta}) P(x^{C_\alpha}, y^{C_\beta})} \quad (*)$$

Considerando que $\Lambda_{x,y} = \Lambda_{x,y}^W \cup \Lambda_{x,y}^{IC}$, donde $\Lambda_{x,y}^W = \{z \in \Lambda_{x,y} | x^C, y^C, z^C\}$ el set intra-cluster de vecinos comunes y su complementario $\Lambda_{x,y}^{IC} = \Lambda_{x,y} \setminus \Lambda_{x,y}^W$ el el set inter-cluster de vecinos comunes.

Así, partiendo de s_{xy} como en (*), con algunos cambios puramente técnicos basados en las probabilidades de tener un conjunto de vecinos dado, condicionado a la pertenencia a cierto cluster⁶⁰ llegamos a que:

$$s_{xy} = \frac{\#\Lambda_{xy}^W}{\#\Lambda_{xy}^{IC}} \approx \frac{\#\Lambda_{xy}^W}{\#\Lambda_{xy}^{IC} + \delta}$$

para cierto $0 < \delta \approx 0$ que previene de una división no definida en los reales.

15.3.13 Total neighbors

El número de vecinos totales se basa en la idea de que, cuanto más conectado está un nodo, más sencillo es que tenga un nuevo link, *independientemente* de la identidad del nodo con el que se relacionaría, pues este último solo aporta, igualmente, el número de vecinos que tiene.

$$s_{xy} = \#\{\Gamma(x) \cup \Gamma(y)\}$$

15.3.14 Otras medidas de similitud

Existen otras medidas de similitud, también implementadas en Neo4j, con la que podemos recoger scores de predicción. Estas medidas no están destinadas a nodos, pero podría llegar a implementarse de alguna manera.

Similitud del coseno

Esta similitud es el coseno del ángulo entre los dos vectores n-dimensionales en el espacio. Si el coseno vale 0 son perfectamente similares y -1 disimilares.

$$s_{xy} = \frac{xy}{||x|| \times ||y||}$$

Similitud de Pearson

⁶⁰Formalmente, $P(\Lambda_{x,y} | x^{C_\alpha}, y^{C_\alpha}) = \frac{|\Lambda_{x,y}^W|}{|\Lambda_{x,y}|}$ y $P(\Lambda_{x,y} | x^{C_\alpha}, y^{C_\beta}) = \frac{|\Lambda_{x,y}^{IC}|}{|\Lambda_{x,y}|}$

La similitud de Pearson es la covarianza de dos vectores n-dimensionales dividida por la desviación típica.

$$s_{xy} = \frac{cov(x, y)}{\sigma_x \sigma_y}$$

Similitud overlap

La similitud overlap mide la superposición entre dos conjuntos.

$$s_{xy} = \frac{\#x \cap y}{\min(\#x, \#y)}$$

15.4 Computación de los algoritmos

En Neo4j están disponibles los siguientes:

- Adamic Adar
- Common Neighbors
- Preferential Attachment
- Resource Allocation
- Same Community (simplemente devuelve si dos nodos están en la misma comunidad)
- Total Neighbors

Mientras, en networkx:

- Resource Allocation
- Jaccard Coefficient
- Adamic Adar
- Preferential Attachment
- Common Neighbors
- Inter-Cluster Common Neighbors
- *Otras medidas de similitud*

Part VIII

Introducción a redes cuánticas

16 Redes cuánticas sociales

16.1 Introducción

Se introduce aquí un acercamiento a través de la física a las redes sociales. En este acercamiento, cada actor se caracteriza por un test sí-no en un sistema físico. Así, se introduce una red social cuántica en la que cada actor i se viene caracterizado por un test de si se presenta o no en el sistema a través de su estado cuántico $|\psi_i\rangle$.

En el estudio tradicional de las redes sociales, éstas vienen descritas por un grafo en cuyos nodos se representan actores y aristas que representan sus interacciones. Sin embargo, estos elementos no capturan la naturaleza de estas interacciones ni explican por qué el elemento i está relacionado o no con otros elementos.

Por ello, se plantea un escenario más general. Se representa cada elemento i por un test sí-no T_i en un sistema físico S con estado inicial ρ y con posibles resultados 1 (sí) o 0 (no).

Puede comprobarse que una red social clásica (CSN) puede caracterizarse en términos de los tests T_i . En una CSN es posible siempre identificar un conjunto mínimo de atributos que describe la presencia o ausencia de un enlace entre dos nodos: cada enlace es descrito por un valor “sí” o “no” para cada atributo.

En esta caracterización de la red social, el test T_i sigue las siguientes reglas: (i) Dos nodos i y j están relacionados si y sólo si existe algún estado ρ para el cual $T_i = T_j = 1$, que simplemente quiere decir que i y j comoparten la propiedad descrita por el atributo ρ . (ii) Si un test T_i es repetido en un mismo estado ρ , debe devolverse siempre un mismo resultado. (iii) Para cualquier ρ , el orden de los tests es irrelevante.

Considérese ahora un escenario aún más general, que llamaremos GSN. Denotamos por $P_\rho(a, b | T_i, T_j)$ la probabilidad conjunta de obtener un resultado $a \in \{0, 1\}$ al realizar el test T_i en el sistema S con estado ρ y obtener un resultado $b \in \{0, 1\}$ al realizar, análogamente, T_j en S para ρ .

El test debe obedecer las siguientes reglas: (i)' $P_\rho(1, 1 | T_i, T_j)$ determina la relación entre i y j : estarán relacionados efectivamente si y sólo si $\exists \rho$:

$P_\rho(1, 1 | T_i, T_j) > 0$. (ii)' $\forall \rho P_\rho(a, a | T_i, T_i) = P_\rho(a | T_i)$, i.e., cuando T_i se repite en S para cierto estado ρ , el resultado debe ser el mismo. (iii)' $\forall \rho P_\rho(a, b | T_i, T_j) = P_\rho(b, a | T_j, T_i)$, el orden es irrelevante.

Una propiedad que consideraremos *deseable* a partir de este momento, que realza la diferencia entre GSN y CSN es la existencia de valores altos de \mathcal{T} , la esperanza de la probabilidad para la cual un nodo i escogido al azar obtiene 1 en el test T_i .

Para las CSN, supongamos que el estado es ρ . Entonces, para un actor i escogido al azar, $\mathcal{T} = \frac{1}{n} \sum_{i=1}^n P_\rho(1 | T_i)$, donde n es el número de elementos. El máximo valor de \mathcal{T} , en cada estado ρ , es el máximo número de elementos compartiendo el valor “sí” para una propiedad (*atributo*) preexistente, dividido por el número de estados. Es decir: $\frac{\omega(G)}{n}$, donde $\omega(G)$ es el número de cliques de G ⁶¹.

Sin embargo, en una GSN, el mayor valor de \mathcal{T} compatible con las reglas (i)', (iii)' es $\frac{\alpha^*(\bar{G})}{n}$, donde \bar{G} denota el complementario de G ⁶² y $\alpha^*(\bar{G})$ se denomina el número de packing fraccional, definido como $\max_{c_j \in \bar{G}} \sum_{i \in V} w_i$, bajo la restricción $\sum_{i \in c_j} w_i \leq 1$.

El punto interesante de esto es que, desde que existen grafos en los cuales $\omega(G) < \alpha^*(\bar{G})$, deben existir redes sociales *usuales* donde \mathcal{T} es superior al máximo valor para CSN.

A partir de este estudio preliminar podemos introducir las redes sociales cuánticas.

16.2 Redes cuánticas sociales

Una red social cuántica (QSN) se define como una red social SN en la que, cada nodo i tiene asociado un estado cuántico $|\psi_i\rangle$. Dichos estados son escogidos para reglejar la naturaleza del grafo de la red en el siguiente sentido. Nodos no-adyacentes (adyacentes) en el grafo corresponden a estados ortogonales⁶³ (no-ortogonales)⁶⁴.

Una QSN puede ser construida a partir de una CSN asignando al nodo i un mecanismo para testear el estado cuántico $|\psi_i\rangle$. La caracterización de las

⁶¹El número de nodos en la clique máxima. Una clique es un subconjunto de nodos del grafo en que cada par de ellos está relacionado por una arista.

⁶²El complementario de G es aquel grafo \bar{G} donde dos nodos son adyacentes si y sólo si no lo son en G .

⁶³Una condición suficiente de ortogonalidad es $\langle \psi_m | \psi_n \rangle = 0$.

⁶⁴Siempre es posible asociar estados cuánticos a los elementos de cualquier red cumpliendo los requisitos de (no) ortogonalidad.

QSN satisfacen las reglas (i)' y (iii)': cada mecanismo recibe un sistema S en un estado cuántico ρ como input y devuelve como output el estado $|\psi_i\rangle$ y el resultado 1, o un estado ortogonal a $|\psi_i\rangle$ y el resultado 0. Estos tests son medidas representadas en mecánica cuántica por proyecciones de rango 1⁶⁵.

En una QSN, \mathcal{T} puede ser mayor que el máximo para otra CSN representada por el mismo grafo. Si cada uno de los nodos reciben un mismo estado $|\Psi\rangle$, de acuerdo a la mecánica cuántica, la probabilidad de obtener 1 cuando se realiza T_i para un cierto i aleatorio es $\mathcal{T} = \frac{1}{n} \sum_{i=1}^n |\langle \Psi | \psi_i \rangle|^2$. Dado G , la cantidad $\frac{1}{n} \max \sum_{i=1}^n |\langle \Psi | \psi_i \rangle|^2$, donde el máximo se toma para todos los vectores $|\Psi\rangle$, $|\psi_i\rangle$ y todas las dimensiones, da el máximo \mathcal{T} para una QSN. Este valor es igual a $\frac{\vartheta(\bar{G})}{n}$, donde $\vartheta(\bar{G})$ es el número de Lovász de \bar{G} ⁶⁶, que puede ser computado con precisión mediante programación semidefinida.

Además, el número de Lovász está acotado inferior y superiormente como sigue:

$$\omega(G) \leq \vartheta(\bar{G}) \leq \chi(G)$$

donde $\omega(G)$ es el número-clique ya descrito y $\chi(G)$ es el número cromático.

El punto es que, para aquellos casos en que $\vartheta(\bar{G}) > \omega(G)$, $\mathcal{T}_{qsn} \geq \mathcal{T}_{csn}$, i.e., el rendimiento de las QSN es superior. Estos casos aumentan conforme se incrementa el número de nodos n del grafo. Tanto es así que para un número suficientemente grande de nodos n^* , la proporción de grafos que cumplen la condición es prácticamente 1.

Una vez que se identifica un grafo en que $\vartheta(\bar{G}) > \omega(G)$, se pueden computar los estados cuánticos $|\Psi\rangle$ y $|\psi_i\rangle$ de mínima dimensión $\xi(G)$ que obtienen la solución óptima, que maximiza \mathcal{T} . El estado $|\Psi\rangle$ es el estado inicial de S necesario

⁶⁵Una proyección es una transformación lineal P tal que $P = P^2$ (1), $P = P^*$ (2). Puede comprobarse que, $\forall q \in \mathbb{C}^n$ $P_q = qq^*$ es una proyección (de rango 1).

⁶⁶Sea $G = (V, E)$ un grafo de n vértices. Un conjunto de vectores unitarios $U = (u_i : i \in V) \subset \mathbb{R}^n$ se denomina una representación ortonormal de G en \mathbb{R}^n si u_i y u_j son ortogonales siempre que i, j no son adyacentes en G .

$$u_i^T u_j = \begin{cases} 1 & i = j \\ 0 & \vec{ij} \notin E \end{cases}$$

Se define el número de Lovász $\vartheta(G)$ como:

$$\vartheta(G) = \min_{c, U} \max_{i \in V} \frac{1}{(c^T u_i)^2}$$

donde c es un vector unitario en \mathbb{R}^n y U es una representación ortonormal de G en \mathbb{R}^n .

El número de Lovász puede entenderse como la cota máxima de la capacidad de Shannon de un grafo y también puede ser obtenido como $\vartheta(G) = \min_A \lambda_{max}(A)$, con A todas las matrices simétricas $n \times n$ tales que $a_{ij} = 1$ si $i = j$ y $a_{ij} = 0$ si $\vec{ij} \notin E$ y $\lambda_{max}(A)$ denota al autovalor máximo de esta matriz.

para obtener la máxima ventaja cuántica.

En la siguiente tabla se estudian las posibles ventajas en todos los grafos isomorfos de 5 a 10 nodos. La ventaja no-cuántica se refiere al número de grafos que obtienen un mejor resultado con CSN que con QSN.

Nodos	Grafos	Ventaja Cuántica	Ventaja No-Cuántica
5	21	1	0
6	112	2	0
7	853	28	0
8	11117	456	0
9	261080	15951	0
10	11716571	957639	4

16.3 Ejemplo de mejora

Cualquier red social a través de internet, como *Facebook* o *Twitter*, tienen un tamaño suficiente como para beneficiarse potencialmente de la asignación de test cuánticos a sus nodos. Un ejemplo es el siguiente: supongamos una compañía que desea vender un producto a tantos usuarios de Facebook como sea posible.

Bajo la asunción de que Facebook es una CSN, la estrategia óptima sería identificar el mayor subgrupo de elementos relacionados mutuamente, obtener sus intereses comunes y diseñar un targeting para estos intereses comunes.

Sin embargo, si Facebook fuera una QSN con exactamente las mismas relaciones que el Facebook real, la compañía tendrá unos mejores resultados si implementa su estrategia de acuerdo a los resultados de los test cuánticos.

Al igual que en una CSN, los nodos de una QSN pueden organizarse en comunidades o clusters. Dado un grafo G , la detección de comunidad será más sencilla en una QSN que en una CSN. La razón es que una QSN con un grafo G dado requiere un sistema físico de dimensión $d_q = \xi(\bar{G})$ ⁶⁷, con $\xi(\bar{G})$ el rango ortogonal del complemento de G . Sin embargo, para crear una CSN se requiere un sistema físico de dimensión $d_c = i(G)$, número de atributos existentes en la red. En la mayoría de los casos se encuentra $d_q < d_c$.

⁶⁷Número de estados perfectamente distinguibles

Part IX

Anti money-laundering

En esta parte se expondrán las principales herramientas de las que dispone la literatura hoy en día para la detección de blanqueo de capitales, especialmente desde el enfoque de redes. El problema a tratar tiene diversos elementos interrelacionados, por lo que se podría llegar a integrar distintos enfoques y fuentes de información con el fin de integrar el conocimiento en un sistema más o menos automatizado, que será el objetivo último de esta parte. Sin embargo, para ello es necesario conocer los métodos utilizados con anterioridad, aunque de manera conceptual, para poder incorporar unos y otros elementos a nuesro sistema de detección, en el que podemos adelantar que tendrá gran peso el análisis de redes.

17 Teoría del AML

En general, la teoría alrededor del AML basado en datamining se puede asociar a una de las líneas de investigación “cerradas” que dominan el campo. Luego, además, hay otros trabajos que tienen una relación menor con los demás, que podríamos considerar independientes. Los trabajos dentro de las líneas usuales suelen seguir un método común, aun con matices. Dichas líneas son presentadas a continuación, con un resumen de las ideas centrales de su metodología compartida.

1. Clustering:

- (a) Zhu (2006) propone comparar las transacciones realizadas frente un grupo de referencia basándose en una distancia a definir y mediante un algoritmo estadístico-geométrico que sigue una idea similar a la del estadístico χ de Pearson.
- (b) Geo Zengan (2009) introduce la idea de la clusterización en dos partes. Primeramente crea comunidades dentro de las observaciones. Una vez dentro de cada una de las comunidades se busca, mediante un algoritmo de detección de outliers, aquellas observaciones más anómalas. Finalmente se calcula un estadístico que combina ambas variables, dando mayor puntuación a aquellas observaciones más atípicas de los clusters más anómalos. La idea es que se *debe eximir*

algunas industrias especiales que, por su naturaleza, no practican ML a pesar de salir como outliers de la clusterización.

- (c) Wang and Dong (2009) incorporan a la estructura anterior la idea de disimilaridad, definida entre dos nodos (aquí de un mismo cluster) como $dis(X, Y) = 1 - \frac{\sum_{i=1}^d \frac{1}{1 + |x_i - y_i|}}{d}$, que utilizan en cuenta del algoritmo de detección de outliers.
- (d) Le Khac and Kechadi (2010) crean un sistema lo suficientemente complejo que se verá en una sección próxima pero, en cuanto al clustering, incorporan variables basadas en la teoría clásica de AML para que el algoritmo se base en ellas como, por ejemplo, $\Delta_2 = |\frac{\beta}{\theta}|_\tau$, el ratio entre el valor de amortización β y el total de las participaciones del inversor θ para cierto periodo de tiempo τ .
- (e) Cao and Do (2012) realizan de nuevo la aproximación en dos partes recurrente en esta línea introduciendo la clusterización mediante CLOPE, que devuelve un histograma con eje-x los miembros del cluster y eje-y la frecuencia de su aparición. A partir de aquí buscan encontrar dos tipos de patrones que, según los autores, son recurrentes en el ML, siendo uno cíclico: $A \rightarrow B \rightarrow C \rightarrow A$ y otro distribuido: $A \rightarrow \{B, C, D\} \rightarrow E$.

2. Rule-based:

- (a) Khan et al. (2013) plantean un modelo de estadística más clásica basado en una red bayesiana: una representación gráfica de una distribución de probabilidad y definen la sospecha en base a reglas del BC de Pakistán basadas en su conocimiento del ML como: “Retirada con frecuencia de grandes cantidades de efectivo por medio de cheques”. A partir de ello computan la probabilidad condicional para los nodos, obteniendo $P(ST|Rn)$, i.e., la probabilidad de un agente de estar haciendo ML con el (in)cumplimiento de ciertas de las reglas.
- (b) Rajput et al. (2014) se basa en las reglas del BC de Pakistán de nuevo para construir una ontología a la que se lanzarán queries, basándose en la idea de monitorización de transacciones independientes.
- (c) Khanuja and Adane (2014) utilizan la teoría de la evidencia de Dempster - Shafer a partir de unas reglas otorgadas por el BC de India para

computar la probabilidad de ML de cada agente. La teoría de la evidencia define tres funciones: bpa o *basic probability assignment*, que especifica la función de masa sobre todos los subconjuntos del espacio probabilístico, la *belief function*, y la *plausibility function* que podemos entender como el “grado de creencia” y “el grado de duda”, respectivamente. Se combinan estas funciones para crear una variable interpretables en términos de sospecha.

- (d) Panigrahi et al. (2009) infiere el nivel de sospecha de acuerdo a la teoría de Shafer, como en el caso anterior. Además, las evidencias que emplean están formadas por tres componentes, uno *rule-based*, otro a través de la propia historia del agente y otro bayesiano, con la idea explicada en el primer trabajo de este apartado. El sistema utilizado, lo suficientemente complejo, será discutido en secciones posteriores.

3. Redes neuronales:

- (a) En Lv, Ji et al. (2008) se propone una red neuronal que parte de un modelo de base radial, que calcula la salida de la función en términos de la distancia a un punto denominado centro, basado en APC-III, algoritmo de clustering para hallar los parámetros de la red oculta. Finalmente, a través de RLS (Mínimos cuadrados recursivos), se actualizan los pesos de las conexiones entre las capas ocultas y de salida.

4. Support Vector Machine (SVM):

- (a) Tang and Yin (2005) sientan las bases de las SVM en el contexto de AML. Dependiendo del espacio (formal) en que se encuentre el dataset y definida la distribución de probabilidad subyacente, podemos llegar a encontrarnos con un conflicto entre una clasificación óptima en el plano y un número mínimo de errores de clasificación. Lo solventan mediante una función de decisión Kernel que incorpora ambos enfoques. A pesar de ser SVM, se trata de unsupervised learning y se llega a una función discriminante $f(x) = sg(\sum_i \alpha_i K(x_i, x) - \rho)$.
- (b) El aporte de Keyan and Tingting (2011) es la adición del cross validation y k-means como método para la búsqueda de los parámetros relevantes en un SVM: el parámetro de sanción y de la función kernel, normalmente c y g . En caso de duda se tomará el par con menor parámetro de sanción.

5. Tree-based:

- (a) Liu et al. (2011) combina BIRCH y k-means para construir el árbol de decisión. BIRCH es un algoritmo de aprendizaje no supervisado que realiza una clusterización jerárquica para crear un árbol en altura (CF tree). Finalmente, usa cierto algoritmo (k-means) para la creación de clusters con todas las hojas como elementos.

6. Otros métodos:

- (a) Lia et al (2008): Se presenta un algoritmo de asignación secuencial que utiliza la detección secuencial de transacciones sospechosas. Este método utiliza dos enfoques complementarios: el historial local de cada cuenta y la información de transacciones con otras cuentas.
- (b) Gao and Xu (2009): Se describe un sistema analista de transacciones con la habilidad de detectar blanqueo de capitales mediante algoritmos como el *transaction mining algorithm* y el *frequent mining algorithm*.
- (c) Zhang and Wang (2010): Se crea una arquitectura para la detección en tiempo real del blanqueo de capitales basada en el análisis de transacciones.
- (d) Michalak and Korczak (2011): Se modelizan las tres partes del blanqueo de capitales: placement, layering & integration de forma de grafo para detectar subgrafos sospechosos.
- (e) Umadevi and Divya (2012): El método consiste en cuatro partes: se contabilizan las transacciones, se ejecuta el algoritmo de frequent pattern minning. Se clusterizan las transacciones y finalmente se ordenan en una tabla según su nivel de sospecha.
- (f) Thangiah et al. (2012): El artículo describe especialmente el marco de detección de blanqueo de capitales en un ambiente virtual con fin de combatir el cibercrimen.
- (g) Mca et al. (2014): Se presenta un acercamiento al AML basado en patrones de comportamiento, especificando métodos de transferencia entre cuentas, cantidad de la transaccion, número de destinatarios y más.
- (h) Paula et al. (2016): Basado en deep learning, se utiliza la metodología CRISP-DM (Cross Industry Standard Process for Data Mining). El

núcleo de este método es el reconocimiento de patrones, donde se utiliza además PCA.

18 Enfoque de redes del AML

En esta sección se explicarán dos artículos existentes sobre el AML desde el enfoque de redes con suficiente detalle. Desde luego, los métodos basados en clusterización, que están diseñados -en principio- para bases de datos relacionales, pueden ser utilizados con bases de datos de grafo como las disponibles y probablemente mejore su resultado. Una vez hecha esta aclaración explicaré los resultados a los que llegan los estudios basados en redes sociales y las principales conclusiones con las que nos debemos quedar. Estos estudios, sin embargo, pueden llegar a pecar de heurística y parten de la observación de los hechos en la realidad, dejando de ser una herramienta puramente estadística. Si se trata de un agravio o una mejoría tendrá que cotejarse con la realidad, al ser *imposible* su comparanza teórica, más allá de cuestiones como la semejanza entre los datos reales de los que disponemos y aquellos que toma el estudio como base, pudiendo llegar a realizar las modificaciones pertinentes con el fin de adaptar el método al caso. Podemos incrustar los algoritmos, así, en una especie de aprendizaje supervisado *humano*.

1. Drezewski et al. (2015): Los autores realizan un estudio empírico que parte de una clusterización, con el fin de encontrar organizaciones subyacentes en la red, y consiguen alcanzar algunos resultados como los que mostraré a continuación, que es lo verdaderamente interesante. A cada nodo, según sus propiedades en la red (betweenness centrality, closeness centrality,...), se le asigna un **rol** dentro de la organización (cluster). Esta es la parte **humana**, pues se toman ciertos roles de acuerdo a definiciones existentes en la realidad de las asignaciones, algo incomprensible para un ordenador. Los roles se dividen en asociados en el largo plazo y en el corto plazo. En el **largo plazo**: *organizers*, constituyen el core de la organización; *insulators*, aíslan el core de la organización, transmiten info en la periferia de la misma, etc.; *communicators* son individuos que controlan la información; *guardians*, controlan quién entra y quién es reclutado; *extenders*, trabajan la extensión de la red; *monitors* se preocupan de la eficiencia de la red; *crossovers*, individuos que trabajan en la organización y en otras áreas. En el **corto plazo** existen *soldiers*, *recruits*, *outliers* y *occasionals*.

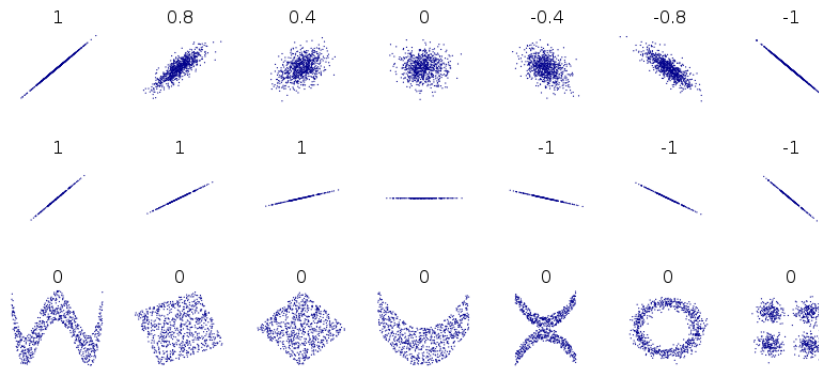
El método, como vemos, trata de asociar a los nodos roles que *se supone* deberían existir en una organización. Para hallar esos roles crean unas reglas sobre lo que entienden que debería aportar cada rol, como se muestra en la siguiente tabla, a modo de ejemplo.

Table 1
Default intervals for the organizer role.

Algorithm	Interval	Interval description
ClosenessC	[0.0, 0.2]	Low since the nodes of this role are separated by the insulators
BetweennessC	[0.0, 0.2]	Low since the nodes of this role do not mediate between other nodes
PageRank	(0.8, 1.0]	High since the nodes of this role are core of the network
Degree	[0.0, 0.2]	Low since the nodes of this role are separated by the insulators
Authoritativeness	(0.8, 1.0]	High since the nodes of this role are the leaders of the network
Hubness	[0.0, 0.2]	Low since the nodes of this role do not point out other nodes with authority

2. Colladon et al (2017) define un método similar, pero más exploratorio-analítico que predictivo. Su objetivo es encontrar qué variables pesan más a la hora de que un nodo sea considerado de alto riesgo. Esto lo consiguen (o esperan conseguirlo) a través de la correlación de Pearson donde, por ejemplo, encuentran que la variable que más influye positivamente al riesgo es el out-degree del sector económico. Por otro lado, el hecho de estar recluido en ese mismo sector (*network constraint economic sector*) influye negativamente. La *closeness* de las transacciones y del área geográfica son las variables que menos importan en este sentido.

El planteamiento tiene un problema claro, que es la baja o nula traslación de los resultados a otro dataset aun semejante. Además, otro problema a tener en cuenta es el uso exclusivo del valor-p, para lo cual sería aconsejable complementar con un análisis del tamaño del efecto; así como el uso exclusivo de la correlación de Pearson y los problemas habituales que sabemos que plantea:



19 Propuestas de integración

19.1 Sistemas de la literatura

En la literatura podemos encontrar diferentes sistemas *complejos* donde se integran algunas de estas técnicas que tienen como fin o, bien conseguir un análisis más global del problema o monitorizar en tiempo real las transacciones con el fin de encontrar aquellas sospechosas.

1. El primer sistema analítico que encontramos es de **Le Khac y Kechady (2010)**. Más allá de la teoría en que se basa su análisis *knowledge-based* (mencionada más arriba), lo interesante de su planteamiento es el sistema, que podemos ver justo debajo.

Como se observa, el sistema tiene dos partes principales: análisis e investigación. En la primera, cada uno de los datasets provenientes de cada uno de los fondos del banco son analizados independientemente, la razón es que **el comportamiento del cliente varía de un fondo a otro dependiendo de las características del mismo**. Por ejemplo, no se comporta igual un cliente en un fondo a corto plazo que en uno a largo. En (1) los datasets son agregados temporalmente por día, mes, año, etc. Esto es algo habitual en la detección de ML. En (2) se definen las variables basadas en ciertas características que *se creen* que pueden ayudar en la clasificación desde el punto de vista de los expertos. Se realiza entonces, en (3) una clusterización (*por una modificación de k-means*) basada en un subconjunto de esos parámetros (*main parameters*) y se realiza una clasificación basada en todos los parámetros en *suspicious* y *unsuspicious*. En este punto se deberá tener un dataset particionado en unos *muy* pocos datos sospechosos y *demasiados* datos no-sospechosos. Por ello, en (4) se utiliza un algoritmo genético para aumentar el ratio en favor de los sospechosos. En (5) se añade una variable temporal, creando un árbol de decisión a través de una red neuronal MLP.

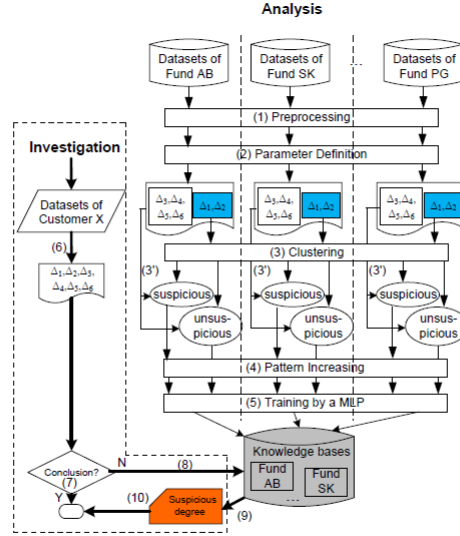
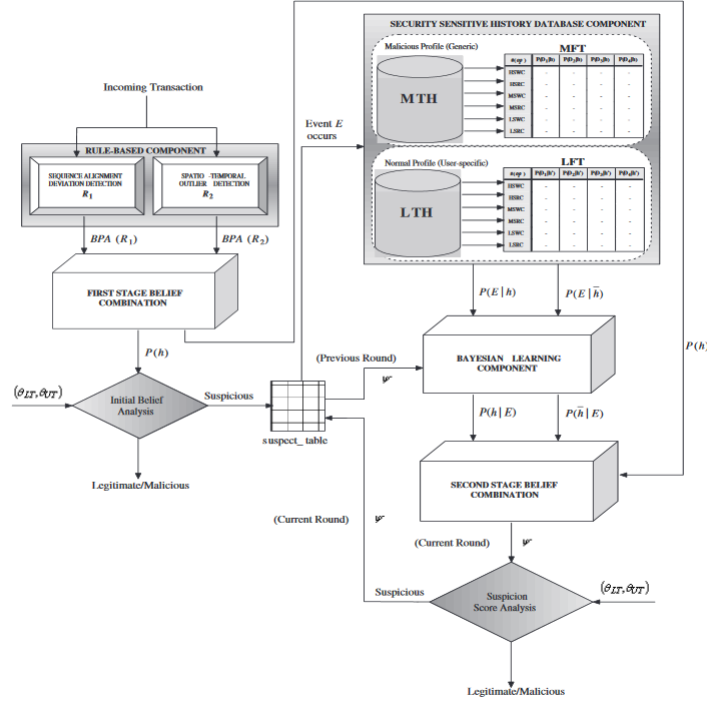


Figure 1. AML analysis and investigation process.

A la hora de investigar los casos nuevos, el sistema plantea lo siguiente: En (6) se calculan los mismos parámetros que anteriormente y en (7) **se pregunta a un experto** si tiene una conclusión acerca de su posible sospecha. Si la respuesta es negativa, se enfrenta al árbol de decisión basado en la red neuronal MLP de (5) y se devuelve un grado de sospecha en (10).

- Otro de los grandes sistemas planteados en la literatura (más allá de aquellos que simplemente monitorizan o clusterizan en dos pasos) es el propuesto en 2009 por Panigrahi.



El sistema en sí no es tan distinto del anterior. Se divide en dos partes, una que comienza con los datos *hasta el momento* que conforma el histórico y otra que parte de una nueva transacción. Si nos fijamos en la primera de las partes mencionadas, el sistema recoge como input dos datasets: LTH y MTH para transacciones históricas *legítimas* y *sospechosas* respectivamente. A partir de aquí se calculan probabilidades condicionales (*bayesian learning*) y métricas de *belief*, ya explicadas anteriormente.

En el otro lado (parte izquierda del sistema), se toman las transacciones entrantes y se pasan por dos filtros basados en reglas lógicas, desde donde se hace un primer análisis: si se puede clasificar como legítimo o malicioso la transacción sale del sistema. Si, por otro lado, pasa a la lista de sospechosos, se enfrenta al modelo entrenado sa base de bayesian learning y belief. Se le da, al final del proceso, un score de sospecha y se acaba determinando si la transacción es legítima o no.

19.2 ¿Qué sistemas se podrían implantar teniendo en cuenta el enfoque de redes?

En este apartado hay que tener en consideración diversas cuestiones. La primera de ellas es la posibilidad de conversión de los nodos a vectores manejables por los algoritmos tradicionales de machine learning a través del denominado proceso *node2vec*, por lo que la distancia entre algoritmos de un tipo y otro, en referencia a algoritmos basados en grafos o en datos estructurados donde emergen los vectores, está salvada y es *sólo* conceptual y, por ello, se tratarán en general en este documento, sin especificar cada caso.

A partir de aquí, podemos tratar de construir un sistema óptimo conforme a los recursos y datos que tengamos disponibles.

Para comenzar, es muy recomendable **una primera clusterización** cuyo fin sea precisar mejor lo que podemos considerar un caso atípico (que, en general, viene asociado a un grado de sospecha mayor) porque, como se explica en [?], *“what is normal behavior for one entity might be ML for other”*. En concreto recomendaría incorporar esta primera clusterización cuando los nodos son empresas o hay nodos que sean empresas, tratando de agruparlas según su industria, mercado, tamaño relativo y demás, que llevan a ciertos patrones de comportamiento similares entre sí, pudiendo llegar posteriormente a medir su disparidad dentro del grupo, una vez eliminado el factor idiosincrático.

Como pudimos comprobar anteriormente, los sistemas muchas veces tienen dos partes: una en que se entrena un modelo con los datos históricos y otra en la que entran transacciones que, en caso de duda, se someten a dicho modelo. Además, hay otros sistemas que directamente monitorizan las transacciones de los usuarios y las comparan tanto histórica como transversalmente. Desde luego, nuestro enfoque va a ser más parecido a este, dado que no queremos que las reglas lógicas entren en el modelo puesto que, por mucho que las refinemos, el sesgo será amplio.

Una evidencia en contra del sistema monitorizado de comparación entre histórico y actual es que los blanqueadores, por lo general, **lo son siempre** y, en cambio, una acción extraña por parte de un usuario (imaginemos la transacción correspondiente a la compra de una casa al contado por una persona normal que realiza la compra para vivir en ese domicilio) se devolvería, con mucha probabilidad, como atípico temporal (si bien es cierto que se podría corregir transversalmente). Estos hechos que podrían suceder deben transformar el modelo en una u otra dirección dependiendo de los agentes que son tratados pero,

en general, personalmente descartaría la temporalidad, sin descartar tomarla en cuenta en base a recurrencias (por ejemplo: el cinco de cada mes se realiza una transacción a un banco panameño).

En un sentido que se puede entrever en el párrafo inmediatamente anterior, los autores suelen mostrar sus preocupaciones acerca del alto nivel de falsos positivos recurrente en los diversos sistemas de AML. Teniendo en cuenta esto y siendo conscientes de las medidas que podemos adoptar, incluida la definición de nuevas medidas de significación (las dos mencionadas, *exigir más* a un positivo para serlo, ya sea haciendo un análisis comparativo de varios métodos o simplemente subiendo el umbral de aceptación en cuanto a grado de sospecha, etc.), debemos trasladar esta idea al analista de la otra parte, léase el banco u otro tipo de organismo, para entender el daño que puede hacer un falso positivo según los pasos de su procedimiento posterior y el coste que tendría tanto dejar un positivo sin sospecha como tratar un falso positivo. El clásico problema de la estadística.

Hay que apuntar que los métodos en sí dentro del sistema no nos preocupan en este primer nivel teórico. Hay casos en que prefiramos, por ejemplo, la clusterización a través de k-means a la detección de comunidades a través del algoritmo de Louvain, dependiendo de los datos (cantidad, número de enlaces, necesidad de restringir al máximo los falsos positivos...) y que deberán realizarse en un nivel empírico con cierto matiz teórico en cada proyecto. Desde luego, también debería probarse distintas alternativas en la fase de detección, ya se escoja una primera clusterización o no. Por ejemplo, se podrían probar cosas *tan dispares* como una red neuronal de grafo GCN, una medida de disparidad local (*dissimilarity*) o un algoritmo de detección de outliers multivariante (*isolation forests*). El output de los mencionados no tiene por qué tener el mismo formato, pero en última instancia debemos ser capaces de clasificar los elementos entre *sospechosos* y *no sospechosos*, aun con la forma -ideal- de grado de sospecha.

Desde luego, para la elección de un algoritmo u otro, la naturaleza de los datos es totalmente relevante. Por ejemplo, no hay posibilidad técnica de realizar un clasificador basado en aprendizaje supervisado basado en datos sin etiquetar y, aunque su contrario es posible, estaríamos de alguna manera perdiendo información. Sin embargo, poniéndonos en este último caso (en el que disponemos de datos etiquetados por los analistas), nuestro objetivo puede ser *doble*: por un lado, lo que parecería más lógico sería realizar aprendizaje supervisado y aprovechar el conocimiento experto de los analistas, replicando su lógica a través de la estadística de manera automática. En caso de éxito, que

suponemos, estaríamos consiguiendo un ahorro enorme de recursos que se podrían destinar a realizar otras operaciones (como investigar profundamente las que el algoritmo señala). Sin embargo, como se ha mencionado, se estaría replicando el modelo seguido hasta ahora. Esto es, aquellos elementos no detectados por el experto hasta ahora de manera estructural no serán detectados tampoco por el nuevo modelo, necesitaríamos **unsupervised learning** para tratar de captar esa nueva información. Por si no ha quedado claro, estos casos son los que se dan cuando existe un tipo de blanqueo de capitales que el experto desconoce profundamente y, aunque investigue su caso suficientemente, no encuentra indicios del mismo.

Desde luego, los expertos remarcan la importancia de conseguir información adicional a la puramente proporcionada por las transacciones-numérica. Tiene bastante recorrido en esta cuestión el PLN. Desde aquel enfoque que (*medio*) rechazamos de monitorización temporal, la información pública (sobre todo en forma noticias, redes sociales) que se incorpore al modelo puede mejorar la clasificación del mismo. Por ejemplo, ciertas transacciones que consideraríamos de probabilidad cero, como comprar en oro la mitad de tu patrimonio, se realizan con frecuencia después de grandes ataques terroristas. Por otro lado, la información proveniente de redes sociales (y de sus mensajes) pueden llegar a *levantar* relaciones entre agentes sobre los que se tiene cierta *sospecha* en el modelo estrictamente numérico, aunque personalmente lo veo poco posible. Desde luego, a nivel interno el poder de esta herramienta no tiene discusión, pudiendo llegar a obtener variables entendibles por el modelo (numéricas) a partir de información descriptiva de la transacción o sus agentes, muy posiblemente a través del *word2vec*, análogo al *node2vec*.

Para finalizar, debemos tener en cuenta también el objetivo del proyecto en el que estemos inmersos. Así, en un proyecto analítico, de apoyo al analista o con fin de tratar de conseguir o perfilar ciertas reglas lógicas, la aproximación a través de las redes sociales que se basa en métricas habituales como la betweenness o el pagerank **no sería tan mala** como parece y su complejidad es mínima. Un paso más a dar sería el uso directo de uno de estos algoritmos presentados, donde yo recomendaría, en general, una primera clusterización con los fines mencionados anteriormente. Este sistema seguiría siendo sencillo pero probablemente lo suficientemente eficaz. Desde luego, la automatización, la (discutible) dimensión temporal y muchos otros de los aspectos comentados, refinan el modelo y son absolutamente necesarios cuando el fin es prácticamente *sustituir* al analista, los datos son suficientemente críticos o la precisión exigida

es máxima.

References

- [1] J. Reichardt, S. Bornholdt. Statistical Mechanics of Community detection. <https://arxiv.org/pdf/cond-mat/0603718.pdf>
- [2] M. Newman. Finding community structure in networks using the eigenvectors of matrices. <https://journals.aps.org/pre/abstract/10.1103/PhysRevE.74.036104>
- [3] A. Clauset, M. Newman, C. Moore. Finding community structure in very large networks. <https://arxiv.org/pdf/cond-mat/0408187.pdf>
- [4] M. Rosvall, C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. <http://arxiv.org/abs/0707.0609>.
- [5] M. Rosvall, D. Axelsson, and C. T. Bergstrom. The map equation. <https://arxiv.org/pdf/0906.1405.pdf>
- [6] U. N. Raghavan, R. Albert, S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. <https://arxiv.org/pdf/0709.2938.pdf>
- [7] VD Blondel, J-L Guillaume, R Lambiotte, E Lefebvre. Fast unfolding of community hierarchies in large networks. <https://arxiv.org/abs/0803.0476>
- [8] M. Girvan, M. E. J. Newman. Community structure in social and biological networks. <https://arxiv.org/pdf/cond-mat/0112110.pdf>
- [9] P. Pons, M. Latapy: Computing communities in large networks using random walks. <https://arxiv.org/pdf/physics/0512106.pdf>
- [10] B. W. Kernighan, S. Lin. An efficient heuristic procedure for partitioning graphs. <https://ieeexplore.ieee.org/document/6771089>
- [11] G. Palla, I. Derenyi, I. Farkas et al.: Uncovering the overlapping community structure of complex networks in nature and society. <http://hal.elte.hu/cfinder/wiki/papers/communitylettm.pdf>

- [12] F. Parés, D. Garcia-Gasulla et al. Fluid Communities: A Competitive and Highly Scalable Community Detection Algorithm. <https://arxiv.org/pdf/1703.09307.pdf>
- [13] A. Galler, J. Fischer: <https://www.enseignement.polytechnique.fr/informatique/ARCHIVES/IF/03/pi/levy2/fischer-galler.pdf>
- [14] R. Tarjan: Depth-First Search and Linear Graph Algorithms. <http://langevin.univ-tln.fr/cours/PAA/extra/Tarjan-1972.pdf>
- [15] T. Schank, D. Wagner: Finding, Counting and Listing all Triangles in Large Graphs, An Experimental Study. https://i11www.iti.kit.edu/extra/publications/sw-fclt-05_t.pdf
- [16] M. Ashtiani: Network Analysis in R, Centrality Measures. <https://www.datacamp.com/community/tutorials/centrality-network-analysis-R>
- [17] Centrality and centrality measures. https://www.ibm.com/support/knowledgecenter/SS3J58_9.0.6/com.ibm.i2.anb.doc/sna_centrality.html
- [18] D. Du: Social Network Analysis: Centrality Measures. http://www2.unb.ca/~ddu/6634/Lecture_notes/Lecture_4_centrality_measure.pdf
- [19] S. Brin, L. Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine <http://infolab.stanford.edu/~backrub/google.html>
- [20] M.E.J. Newman: The mathematics of networks. <http://www-personal.umich.edu/~mejn/papers/palgrave.pdf>
- [21] J. Kleinberg: Authoritative Sources in a Hyperlinked Environment. <https://www.cs.cornell.edu/home/kleinber/auth.pdf>
- [22] J. Zhang, D. Chen, Q. Dong: Identifying a set of influential spreaders in complex networks. <https://www.nature.com/articles/srep27823.pdf>
- [23] P. Boldi, S. Vigna: Axioms for Centrality. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.768.2652&rep=rep1&type=pdf>
- [24] J. Li, P. Willett: ArticleRank: a PageRank-based Alternative to Numbers of Citations for Analysing Citation Networks. <https://pdfs.semanticscholar.org/3940/>

300f122fad2144921bb5c41cc365f9e6406e.pdf?_ga=2.211708624.
430056176.1562926157-1174962054.1560510108

- [25] U. Brandes, D. Fleischer: Centrality Measures Based on Current Flow. <https://link.springer.com/content/pdf/10.1007%2Fb106485.pdf>
- [26] E. Estrada, D. J. Highman, N. Hatano: Communicability betweenness in Complex Networks. <https://arxiv.org/abs/0905.4102>
- [27] M. E. J. Newman: Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. <http://journals.aps.org/pre/abstract/10.1103/PhysRevE.64.016132>
- [28] K. Goh, B. Kahng, D. Kim: Universal behavior of Load Distribution in Scale-Free Networks. <http://phya.snu.ac.kr/~dkim/PRL87278701.pdf>
- [29] M. Piraveenan, M. Prokopenko, L. Hossein: Percolation Centrality: Quantifying Graph-Theoretic Impact of Nodes during Percolation in Networks. <https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0053095&type=printable>
- [30] A. M. Kermarrec, E. Le Merrer, B. Sericola: Second order centrality. <https://www.sciencedirect.com/science/article/pii/S0140366410002689>
- [31] E. Mones, L. Vicsek, T. Vicsek: Hierarchy Measure for Complex Networks <https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0033799&type=printable>
- [32] E. Estrada, J. A. Rodríguez-Velázquez: Subgraph Centrality in Complex Networks. <https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0033799&type=printable>
- [33] E. Estrada: Characterization of 3D molecular structure. <https://www.sciencedirect.com/science/article/pii/S0009261400001585>
- [34] N. Garg, B. Favre, K. Reidhammer: ClusterRank, a Graph Based Method for Meeting Summarization https://www.researchgate.net/publication/221478500_ClusterRank_A_Graph_Based_Method_for_Meeting_Summarization/link/0c9605171c97f02ec1000000/download

- [35] D. Shah, T. Zaman: Finding Rumor Sources on Random Graphs. https://pdfs.semanticscholar.org/b922/a43d864b6e692d3ca8a7166a9c267e08f7af.pdf?_ga=2.242896130.1863287067.1563184058-1174962054.1560510108
- [36] R. Burt: Structural Holes: The Social Structure of Competition. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1496205
- [37] S. Borgatti: Structural Holes, Unpacking Burt's Redundancy Measures. [http://www.analytictech.com/connections/v20\(1\)/holes.htm](http://www.analytictech.com/connections/v20(1)/holes.htm)
- [38] R. E. Tarjan: A note on finding the bridges of a graph. <https://www.sciencedirect.com/science/article/pii/0020019074900039?via%3Dihub>
- [39] J. M. Schmidt: A Simple Test on 2-Vertex and 2-Edge Connectivity. <https://doi.org/10.1016%2Fj.ipl.2013.01.016>
- [40] H. Scarf, L. Shapley: On cores and indivisibility. <https://doi.org/10.1016%2F0304-4068%2874%2990033-0>
- [41] P. Milgrom, I. Segal: Deferred-Acceptance Auctions and Radio Spectrum Reallocation. <http://web.stanford.edu/~isegal/heuristic.pdf>
- [42] P. J. Kelly: A congruence theorem for trees. <https://projecteuclid.org/euclid.pjm/1103043674>
- [43] J. Barát, G. Joret, D. R. Wood: Disproof of the List Hadwiger Conjecture. <https://www.combinatorics.org/ojs/index.php/eljc/article/view/v18i1p232/pdf>
- [44] C. Couto, P. J. de Rezende, C. C. de Souza: An exact algorithm for minimizing vertex guards on art galleries. <https://onlinelibrary.wiley.com/doi/full/10.1111/j.1475-3995.2011.00804.x>
- [45] H. Weiss: The SIR model and the Foundations of Public Health. <http://mat.uab.cat/matmat/PDFv2013/v2013n03.pdf>
- [46] B. V. Cherkassky, A. V. Goldberg, T. Radzik: Shortest paths algorithms. Theory and experimental evaluation. <http://ftp.cs.stanford.edu/cs/theory/pub/goldberg/sp-alg.ps.Z>

- [47] J. Byrka, F. Grandoni, T. Rothvoss: An improved LP-based Approximation for Steiner Tree. <https://dl.acm.org/citation.cfm?id=1806769>
- [48] M. Held, R. M. Karp: A dynamic Programming Approach to Sequencing Problems. <https://epubs.siam.org/doi/10.1137/0110015>
- [49] M. Dorigo, L. M. Gambardella: Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. <http://people.idsia.ch/~luca/acs-ec97.pdf>
- [50] B. Chen, M. Matsumoto, J. Wang: A short proof of Nash-Williams theorem for arboricity of a graph. <https://link.springer.com/article/10.1007%2F01202467#citeas>
- [51] H. N. Gabow, H. H. Westermann: Forests, frames, and games: Algorithms for matroid sums and applications. <https://link.springer.com/article/10.1007%2F01758774>
- [52] V. M. Malhortra, M. P. Kumar, S. N. Maheshwari. An $\mathcal{O}(|V|^3)$ algorithm for finding maximum flows in networks. <https://www.sciencedirect.com/science/article/pii/0020019078900169?via%3Dihub>
- [53] J. B. Orlin: Max flows in $\mathcal{O}(nm)$ time, or better. https://dspace.mit.edu/bitstream/handle/1721.1/88020/Orlin_0%28nm%29MaxFlow.pdf?sequence=1&isAllowed=y
- [54] R. Z. Hwang, R. C T. Lee, R. C. Chang: The slab dividing approach to solve the EuclideanP-Center problem. <https://link.springer.com/article/10.1007%2F01185335>
- [55] M. Badoiu, S. Har-Peled, P. Indyk: Approximate Clustering via Core-Sets. <https://www2.cs.duke.edu/courses/spring07/cps296.2/papers/badoiu02approximate.pdf>
- [56] T. F. Gonzalez: Clustering to minimize the maximum intercluster distance. <https://web.archive.org/web/20130124011012/http://www.cs.ucsb.edu/~TEO/papers/Ktmm.pdf>
- [57] M. Needham: Link Prediction with Neo4j. <https://medium.com/neo4j/link-prediction-with-neo4j-part-1-an-introduction-713aa779fd9>
- [58] D. Liben-Nowell, J. Kleinberg: The Link Prediction Problem for Social Networks. <https://www.cs.cornell.edu/home/kleinber/link-pred.pdf?>

- [59] S. Soundarajan, J. Hopcroft: Using Community Information to Improve the Precision of Link Prediction Methods. http://delivery.acm.org/10.1145/2190000/2188150/p607-soundarajan.pdf?ip=150.244.20.59&id=2188150&acc=ACTIVE%20SERVICE&key=E16C5911FC136D5E%2EE16C5911FC136D5E%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1563526743_b2c5c5e27950bb801361ef3a0e245b34
- [60] J. C. Valverde-Rebaza, A. de Andrade: Link prediction in complex networks based on cluster information. (pp 92-101) <https://link.springer.com/content/pdf/10.1007%2F978-3-642-34459-6.pdf>
- [61] K. M. Carley: Dynamic Network Analysis. http://www.casos.cs.cmu.edu/projects/book/DNA-Book_Draft.pdf
- [62] C. Bisconti, A. Corallo, M. de Maggio: Quantum Modeling of Social Networks (pp 70-77) <https://link.springer.com/content/pdf/10.1007%2F978-3-642-04754-1.pdf>
- [63] A. Cabello, L. E. Danielsen, A. J. López-Tarrida: Quantum Social Networks. <https://arxiv.org/pdf/1112.0617.pdf>
- [64] E. W. Wiesstein: Lovász Number. <http://mathworld.wolfram.com/LovaszNumber.html>
- [65] R. C. Read, R. J. Wilson: An Atlas of Graphs. <https://pdfs.semanticscholar.org/4fa2/b8abcd2a2ad5562e4f8c964742c2106cfb58.pdf>
- [66] S. N. Dorogovste, A. V. Goltsev, J. F. F. Mendes: Pseudofractal Scale-free Web. <https://arxiv.org/pdf/cond-mat/0112143.pdf>
- [67] J. C. Miller: Percolation and epidemics in random clustered networks. https://scholar.harvard.edu/files/joelmiller/files/random_clustered_pre.pdf
- [68] H. Whitney: Congruent Graphs and the Connectivity of Graphs. <https://www.jstor.org/stable/pdf/2371086.pdf?refreqid=excelsior%3A0a9bc9e5eb129895a2dd4444de12cf55>
- [69] W. W. Zachary: An Information Flow Model for Conflict and Fission in Small Groups. <http://www1.ind.ku.dk/complexLearning/zachary1977.pdf>

- [70] A. Davis, B. B. Gardner: Deep South; a social anthropological study of caste and class. <https://psycnet.apa.org/record/1942-00249-000>
- [71] R. L. Breiger, P. E. Pattison: Cumulated Social Roles; The duality of persons and their algebras. <https://www.sciencedirect.com/science/article/pii/0378873386900067>
- [72] D. E. Knuth: The Stanford GraphBase; A Platform for Combinatorial Computing. <https://pdfs.semanticscholar.org/b9dc/985b40ab9146093e7088f012db99e8d7172b.pdf>
- [73] L. Baldesi, A. Markopoulou, C. T. Butts: Spectral Graph Forge; Graph Generation Targeting Modularity. <https://arxiv.org/abs/1801.01715>
- [74] X. Ying, X. Wu: On Randomness Measures for Social Networks. <https://epubs.siam.org/doi/pdf/10.1137/1.9781611972795.61>
- [75] T. Zhu: An Outlier Detection Model Based on Cross Dataset Comparison for Financial Surveillance. <https://ieeexplore.ieee.org/document/4041296>
- [76] Z. Gao: Application of Cluster-Based Local Outlier Factor Algorithm in Anti-Money Laundering. <https://ieeexplore.ieee.org/document/5302396>
- [77] X. Wang, G. Dong: Research on Money Laundering Detection Based on Improved Minimum Spanning Tree and Its Application. <https://ieeexplore.ieee.org/document/5362309>
- [78] N. Le-Khac, T. Kechadi: Application of Data Mining for Anti-Money Laundering Detection: A Case Study. https://www.researchgate.net/publication/220765959_Application_of_Data_Mining_for_Anti-money_Laundering_Detection_A_Case_Study
- [79] D. K. Cao, P. Do: Applying Data mining in Money Laundering Detection for the Vietnamese Banking Industry. <https://link.springer.com/content/pdf/10.1007%2F978-3-642-28490-8.pdf>
- [80] N. Khan, A. S. Larik et al.: A Bayesian Approach for Suspicious Financial Activity Reporting. https://www.researchgate.net/publication/274576393_A_bayesian_approach_for_suspicious_financial_activity_reporting

- [81] Q. Rajput, N. Khan, S. Haider: Ontology Based Expert-System for Suspicious Transactions Detection. https://www.researchgate.net/publication/269847173_Ontology_Based_Expert-System_for_Suspicious_Transactions_Detection
- [82] H. K. Khanuja, D. S. Adane: Forensic Analysis for Database Transactions. https://link.springer.com/chapter/10.1007/978-3-662-44966-0_19
- [83] S. Panigrahi, S. Sural, A. K. Majumdar: Two-stage Database Intrusion Detection by Comining Multiple Evidence and Belief Update. <https://link.springer.com/article/10.1007/s10796-010-9252-2>
- [84] M. Weber, J. Chen et al.: Scalable Graph Learning for Anti-Money Laundering: A First Look. <https://arxiv.org/pdf/1812.00076.pdf>
- [85] R. Drezewski, J. Sepielak, W. Filipkowski: The application of social network analysis algorithms in a system supporting money laundering detection. <https://www.sciencedirect.com/science/article/pii/S0020025514009979>
- [86] A. F. Colladon, E. Remondi: Using Social Network Analysis to Prevent Money Laundering. <https://www.sciencedirect.com/science/article/pii/S0957417416305139>
- [87] L. T. Lv, N. Ji, J. L. Zhang: A RBF Neural Network Model for Anti-Money Laundering. <https://ieeexplore.ieee.org/abstract/document/4635778>
- [88] J. Tang, J. Yin: Developing an Intelligent Data Discriminating System of Anti-Money Laundering Based on SVM. <https://ieeexplore.ieee.org/abstract/document/1527539>
- [89] L. Keyan, Y. Tingting: An Improved Support-Vector Network Model for Anti-Money Laundering. <https://ieeexplore.ieee.org/document/6092658>
- [90] R. Liu, X. L. Qian et al.: Research on Anti-Money Laundering Based on Core Decision Tree Algorithm. <https://ieeexplore.ieee.org/document/5968986>

- [91] X. Liu, P. Zhang, D. Zeng: Sequence Matching for Suspicious Activity Detection in Anti-Money Laundering. https://link.springer.com/chapter/10.1007/978-3-540-69304-8_6
- [92] S. Gao, D. Xu: Conceptual Modeling and Development of an Intelligent Agent-Assisted Decision Support System for Anti-Money Laundering. <https://www.sciencedirect.com/science/article/pii/S0957417407005891>
- [93] C. Zhang, Y. Wang: Research on Application of Distributed Data Mining in Anti-Money Laundering Monitoring System. <https://ieeexplore.ieee.org/document/5487272>
- [94] K. Michalak, J. Korczak: Graph Mining Approach to Suspicious Transaction Detection. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.232.376&rep=rep1&type=pdf>
- [95] M. Thangiah, S. Basri: A Framework to Detect Cybercrime in the Virtual Environment. <https://ieeexplore.ieee.org/abstract/document/6297307>
- [96] G. Krishnapriya, M. Phil et al.: Money Laundering Analysis Based on Time Variant Behavioral Transaction Patterns Using Data Mining. <https://pdfs.semanticscholar.org/8cd1/b0b6557b7fc8f2d9c7e04c7afc44f67a6297.pdf>
- [97] E. L. Paula, M. Ladeira et al.: Deep Learning Anomaly Detection as Support Fraud Investigation in Brazilian Exports and Anti-Money Laundering. <https://ieeexplore.ieee.org/document/7838276>