



ugr | Universidad
de **Granada**

PROYECTO FIN DE CARRERA
INGENIERÍA INFORMÁTICA

Detección de Comunidades en Redes Complejas

Aplicación en Redes Sociales

Autores

Aarón Rosas Rodríguez
Francisco Javier Gijón Moleón

Director

Fernando Berzal Galiano



Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Granada, Septiembre 2014

Detección de Comunidades en Redes Complejas

Aplicación en Redes Sociales

Aarón Rosas Rodríguez

Francisco Javier Gijón Moleón

Palabras clave: clustering, redes complejas, grafos, detección de comunidades, espectral, modularidad, jerárquico, dendograma, particional, red social, subgrafo

Resumen

El modelo de red para representar sistemas complejos es una práctica muy habitual en diversos campos de la investigación científica. Es la forma de representar un conjunto de elementos o entidades que interaccionan, compar-ten o se relacionan entre sí.

En este tipo de sistemas, es muy común que se den conjuntos o grupos de elementos que se encuentran altamente conectados unos con otros, dando lugar a una estructura de comunidad.

Estas comunidades, puede ser interesante identificarlas bien para extraer conocimiento o permitir un mayor control del sistema. El reto que nos plan-teamos en nuestro Proyecto Fin de Carrera es el de investigar, comprender e implementar técnicas de las más famosas que tratan de resolver esta meta y darle una aplicación en el ámbito de las Redes Sociales que tan presentes están hoy día en nuestras vidas.

Community Detection in Complex Networks

Social Networks Application

Aarón Rosas Rodríguez

Francisco Javier Gijón Moleón

Keywords: clustering, complex networks, graph, community detection, spectral, hierarchical, dendrogram, partitional, social network, subgraph

Resumen

The network model to represent complex systems is a common practice in various fields of scientific research. It is the way to represent a set of elements or entities that interact, share and relate to each other.

In such systems, it is common sets or groups of elements that are highly connected with each other, forming a structure of community.

These communities may be interesting to identify. The challenge we set our Final Degree Project is to investigate, understand and implement techniques of the most famous who try to meet this goal and give an application in the field of Social Networks as these are today in our lives .

Yo, **Aarón Rosas Rodríguez**, alumno de la titulación INGENIERÍA INFORMÁTICA de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada, con DNI 75139516B, autorizo la ubicación de la siguiente copia de mi Proyecto Fin de Carrera en la biblioteca del centro para que pueda ser consultado por las personas que lo deseen.

Fdo.: Aarón Rosas Rodríguez.

Yo, **Francisco Javier Gijón Moleón**, alumno de la titulación INGENIERÍA INFORMÁTICA de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada, con DNI 76626706K, autorizo la ubicación de la siguiente copia de mi Proyecto Fin de Carrera en la biblioteca del centro para que pueda ser consultado por las personas que lo deseen.

Fdo.: Francisco Javier Gijón Moleón.

D. **Fernando Berzal Galiano**, Profesor del Departamento CCIA de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado **Detección de Comunidades en Redes Complejas, Aplicación en Redes Sociales**, ha sido realizado bajo su supervisión por **Aarón Rosas Rodríguez** y **Francisco Javier Gijón Moleón**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada, Septiembre de 2014.

El director: **Fernando Berzal Galiano**.

Agradecimientos

A nuestra familia y amigos.
A nuestro tutor, por su apoyo y consejos.

*Las matemáticas son el alfabeto
con el cual Dios ha escrito el Universo.*

Galileo Galilei.

ÍNDICE

1.	Introducción	11
1.1.	Motivación.....	11
1.2.	Objetivos	12
1.3.	Organización del trabajo	12
2.	Redes y Comunidades	15
2.1.	Red Compleja.....	15
2.2.	Grafos.....	16
2.2.1.	Camino	16
2.2.2.	Adyacencia	16
2.2.3.	Peso.....	17
2.2.4.	Grado	17
2.2.5.	Representación Matricial	18
2.2.5.1.	Matriz de adyacencia	18
2.2.5.2.	Matriz de incidencia.....	18
2.2.6.	Subgrafos	18
2.3.	Comunidades.....	20
2.3.1.	Análisis local	21
2.3.2.	Análisis global.....	21
2.4.	Medidas.....	22
2.4.1.	Distancia.....	22
2.4.2.	Densidad	22
2.4.3.	Centralidad.....	23
2.4.3.1.	Centralidad por Grado	23
2.4.3.2.	Centralidad por Cercanía	24
2.4.3.3.	Centralidad por Vector Propio	24
2.4.3.4.	Centralidad por Intermediación	25
2.4.4.	Coeficiente de Agrupamiento	25
2.4.5.	Modularidad.....	27
3.	Detección de Comunidades.....	29
3.1.	Métodos Particionales	29
3.1.1.	Particionamiento sobre un Espacio Métrico	29
3.1.1.1.	k-Means	30
3.1.2.	Particionamiento sobre grafos.....	32
3.1.2.1.	Corte Mínimo o Min Cut	32
3.1.2.2.	Corte Mínimo Normalizado o NCut.....	33
3.1.2.3.	Mínimo Radio de Corte o RadioC.....	33
3.1.2.4.	Corte Min-Max o Min-Max Cut	33
3.1.2.5.	Kernighan-Lin	34
3.2.	Método Jerárquicos	36
3.2.1.	Métodos divisivos	37
3.2.1.1.	Método de Newman y Girvan	37
3.2.1.2.	Método de Radicchi	39
3.2.2.	Métodos aglomerativos	40
3.2.2.1.	Simple-Link.....	40

3.2.2.2.	Complete-Link	41
3.2.2.3.	Average-Link.....	41
3.3.	Métodos Greedy basados en modularidad	42
3.3.1.	Método Fast Greedy	42
3.3.2.	Multi Step Greedy	43
3.4.	Métodos Espectrales.....	44
3.4.1.	Laplaciano de un grafo	45
3.4.2.	Estimación del número de clusters	46
3.4.3.	Algoritmo genérico de particionamiento espectral	46
3.4.4.	Bisección Espectral o EIG1	47
3.4.5.	UKMeans.....	47
3.4.6.	Algoritmo NJW o KNSC1.....	48
3.5.	Evaluación de los resultados	49
3.5.1.	Modularidad.....	49
3.5.2.	Coeficiente de Cohesión	49
3.5.3.	Coeficiente de Separación	50
3.5.4.	Coeficiente de Silueta	51
3.5.5.	Coeficiente de Cobertura	51
3.6.	Diseño de la implementación.....	52
3.6.1.	Algoritmos.....	52
3.6.2.	Medidas	54
4.	Implementación.....	56
4.1.	Paralelismo	56
5.	Pruebas.....	58
5.1.	Pruebas de Calidad	58
5.2.	Pruebas de Tiempo	67
6.	Aplicaciones en Redes Sociales	70
7.	Wisper	73
7.1.	Una herramienta común para diferentes Redes Sociales	73
7.2.	¿Para qué sirve?.....	73
7.3.	¿Cómo funciona?	74
7.4.	Trabajando con Twitter.....	75
7.4.1.	La red de nuestro Timeline	75
7.5.	Diseño e implementación.....	77
7.5.1.	Interfaz de usuario	78
7.5.2.	Servidor Web	79
7.5.3.	Web Service	80
7.5.3.1.	Elección del Algoritmo.	80
7.6.	Manual de Uso.....	86
8.	Conclusiones y Trabajos Futuros	91
9.	Bibliografía y listados.....	93

1. Introducción

1.1. Motivación

El uso de redes para modelar sistemas complejos es una práctica muy común en multitud de campos de investigación. Son extremadamente útiles para representar, en forma de red, un conjunto de elementos que interactúan entre sí, como por ejemplo la interacción entre proteínas o genes, correlaciones entre precios de acciones bursátiles, intercambio de información en internet o redes sociales. Al estudiar la estructura generada por la red y analizar cómo interactúan los distintos elementos, podemos llegar a comprender mejor el comportamiento del sistema.

Es frecuente que, en este tipo de sistemas, se formen grupos de elementos altamente conectados entre sí, lo que se conoce como estructura de comunidad, y es el reto que plantea detectar este tipo de situaciones lo que ha hecho que, al igual que otras áreas como la biología o la sociología, las ciencias de la computación también se hayan interesado por él, lo que ha permitido el nacimiento de varias familias de algoritmos que abordan y resuelven, con mayor o menor acierto, este problema, teniendo en cuenta el grado de dificultad computacional que plantea. Estos algoritmos pertenecen a la rama de Aprendizaje No Supervisado, concretamente son Algoritmos de Agrupamiento o Clustering.

Existen varias familias de algoritmos atendiendo a su forma de actuar, normalmente inspirados en algoritmos clásicos de Clustering. Una de las familias más importantes son Algoritmos Jerárquicos cuya finalidad se centra en construir el dendograma de la red bien comenzando por la raíz y dividiendo clusters (Métodos Divisivos) o comenzando desde las hojas y agrupando clusters (Métodos Aglomerativos). Otra familia son los Algoritmos Modulares que siguen estrategias voraces para optimizar una medida de calidad: la modularidad general de la red. También tenemos la familia de los Algoritmos Particionales que trabajan tratando a los nodos como puntos en el espacio que posteriormente es seccionado formando los clusters ó aplicando técnicas de corte mínimo en grafos. Por último, la familia de los Algoritmos Espectrales, que se apoyan en el cálculo de autovalores y autovectores para transformar el conjunto de nodos de la red en un conjunto de puntos del espacio métrico, susceptibles de ser agrupados mediante alguna método de clustering clásico.

Todas estas familias comparten el mismo problema y es que no son capaces de detectar solapamiento entre las comunidades aunque en algunas ocasiones esto puede existir. Un ejemplo muy claro se encuentra en las Redes

Sociales cuando algunas personas pueden pertenecer, por ejemplo, al grupo de familiares, amigos o compañeros de universidad a la vez. Para suplir esta carencia se diseñaron técnicas más modernas como son los Métodos de Percolación de Cliques que buscan subconjuntos adyacentes, es decir, conjuntos de nodos que comparten al menos uno de ellos. El problema es que son algoritmos computacionalmente muy costosos lo que los hace poco viables en cuanto la red crece, sobre todos con el número de conexiones.

Nuestro trabajo se ha centrado en estudiar las diferentes técnicas que se nos plantean, medidas para la evaluación de los resultados y, como caso práctico, hemos aplicado la teoría sobre una red compleja muy conocida: **Red Social Twitter**.

1.2. Objetivos

Los objetivos fijados para este trabajo fin de carrera son:

- Comprender los conceptos relacionados con las redes complejas y la detección de comunidades en ellas.
- Analizar e implementar las diferentes técnicas que se nos brindan para resolver el problema de la Detección de Comunidades, puntos fuertes y carencias de cada una de ellas.
- Analizar e implementar las diferentes medidas de evaluación no supervisada de los resultados de los algoritmos.
- Realizar un caso práctico sobre una red compleja real, en nuestro caso *Twitter*.

1.3. Organización del trabajo

En el capítulo 2 se hace una revisión de los conceptos previos necesarios para poder abordar el problema. Se indican conceptos básicos sobre teoría de grafos, se comentan los diferentes tipos de subgrafos que podemos hacer corresponder con el término de comunidad y diferentes medidas que exaltan las características propias de la definición de comunidad.

En el capítulo 3 se hace el estudio revisión de los principales métodos de Detección de comunidades sin solapamiento. Clasificaremos dichos métodos en base a medidas utilizadas o a la forma de actuar. Estableceremos una serie de medidas que nos ayudarán a evaluar los resultados obtenidos en función de la calidad de los clusters generados cuantificando las características descritas en el capítulo 2.

En el capítulo 4 introduciremos información sobre la implementación de los algoritmos, detalles de la paralelización de los mismos y del repositorio donde está alojado el código fuente.

En el capítulo 5 se presentaran los resultados obtenidos en las diferentes pruebas ejecutadas con redes clásicas de ejemplo y redes reales extraídas de nuestra propia red social de *Facebook*, donde mostraremos una serie de graficas y tablas con los resultados obtenidos.

En el capítulo 6 se plantea un caso real de red compleja donde se pretende aplicar la teoría vista en los diferentes apartados, concretamente sobre el caso de las Redes Sociales, cuyo auge en los últimos años ha hecho crecer el campo de estudio que aquí tratamos. Veremos diferentes redes sociales estableciendo una serie de pros y contras entre las más populares, decantándonos por *Twitter*.

En el capítulo 7 presentaremos una aplicación real de Detección de comunidades aplicada sobre la red social *Twitter*, que nos permitirá visualizar nuestro Timeline de una forma diferente. Mediante esta aplicación tendremos la posibilidad de agrupar nuestros *tweets* para formar conversaciones y áreas temáticas (Política, prensa, deportes, etc..).

En el capítulo 8 expondremos las conclusiones obtenidas en el desarrollo de nuestro trabajo e indicaremos las posibles ampliaciones del mismo, así como futuras mejoras que propondremos para nuestra aplicación.

2. Redes y Comunidades

Si nos paramos a pensar un momento podremos ver que estamos rodeados de sistemas complejos que están formados por elementos que interaccionan entre si y que, a su vez, pueden ser sistemas complejos como por ejemplo Internet como interconexión de routers que a su vez interconectan diferentes dispositivos o la red de carreteras que unen países entre si, compuestos por ciudades unidas a su vez, pueblos, barrios, etc. Para poder comprender estos sistemas necesitamos una forma de modelar los elementos que los componen y sus interacciones. En este contexto aparecen las redes como una herramienta imprescindible para el análisis de estos sistemas ya que nos permiten representar, de forma sencilla, cualquier conjunto de elementos que interactúan entre sí.

2.1. Red Compleja

[Palla et al. \(2005\)](#) se referían a una red compleja como una red que no tiene una topología trivial, es decir, que tienen una serie de características que no se esperan en redes aleatorias como un alto coeficiente de agrupamiento, evidentes signos de estructura jerarquizada además de una clara estructura comunitaria. Como ejemplo, podemos ver la imagen de la red de transporte aéreo mundial. Cada enlace, en gris, representa el tráfico de pasajeros entre más de 1.000 aeropuertos del planeta, la red completa tiene más de 35.000 enlaces. Las líneas rojas representan el esqueleto de la red, una estructura arbórea de solo 1.300 enlaces, que son las conexiones más importantes de la red.

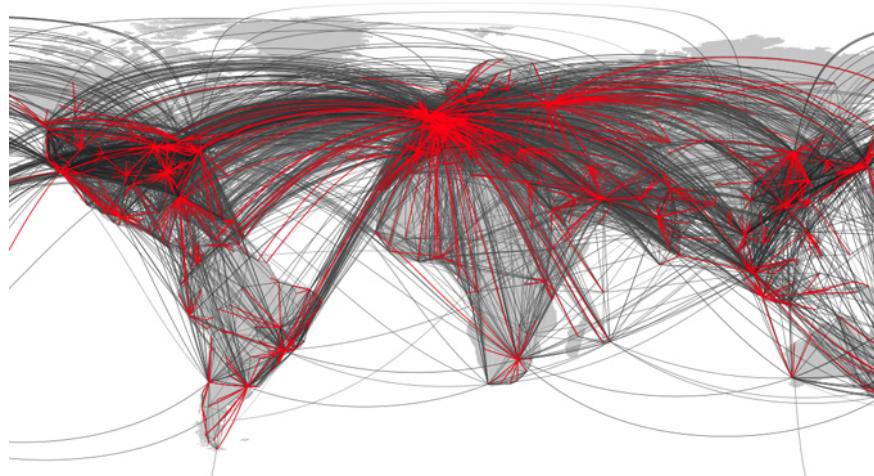


Ilustración 1. Transporte aéreo (Fuente: Northwestern University)

2.2. Grafos

[Newman \(2010\)](#) plantea usar la definición más básica de red conocida como **grafo**. Aunque usar este tipo de modelo puede suponer la perdida de información, reducir un sistema complejo a una red sencilla puede facilitar mucho el proceso de análisis.

Un **grafo** es un par $G = (V, E)$ donde V es un conjunto (normalmente finito) de nodos o vértices y E es un conjunto de aristas o arcos que relacionan estos nodos. Dependiendo de la dirección de estas aristas (si la relación es en ambas direcciones o solo en una) tendremos:

- Un **grafo no dirigido** donde E es un conjunto de pares no ordenados de elementos distintos de V . Un par no ordenado es un conjunto de la forma $\{a, b\}$ donde $\{a, b\} = \{b, a\}$.
- Un **grafo dirigido** donde E es un conjunto de pares ordenados de elementos distintos de V . Un par ordenado es un conjunto de la forma $\{a, b\}$ donde a es el elemento origen y b el elemento destino.

Las aristas que unen los nodos consigo mismos las denotaremos **lazos**.

A partir de aquí, trataremos con grafos no dirigidos con un conjunto finito de elementos, por lo que por simplicidad los denotaremos como **grafo**.

2.2.1. Camino

El **camino** entre un par de nodos es la secuencia de aristas consecutivas que los conectan. La longitud de dicho camino será el número de aristas que lo componen. Denotamos al camino como:

- **Simple** cuando no emplea más de una vez la misma arista.
- **Compuesto** cuando se emplea alguna arista 2 veces o más, aunque no de forma consecutiva.

Al camino que comienza y acaba en el mismo vértice lo denotaremos como **ciclo**.

2.2.2. Adyacencia

Sea $G = (V, E)$ un grafo:

Se dice que dos nodos a y b son **adyacentes** si existe una arista entre ellos, es decir, si $\{a, b\} \in E$. A este par de nodos también se les llama **nodos vecinos**.

Se dice que una arista es **incidente** a un nodo si esta lo une a otro nodo, por ejemplo $\{a, b\} \in E$ es incidente al nodo a e incidente al nodo b .

De igual forma podemos definir a dos aristas como adyacentes si son **incidentes** en el mismo nodo, por ejemplo $\{a, b\} \in E$ y $\{a, c\} \in E$ son adyacentes.

2.2.3. Peso

A veces nos interesa enriquecer nuestra red mediante un valor, peso o fuerza que represente la fuerza de las relaciones entre los elementos. Por ejemplo, en una red de carreteras, el valor puede significar la distancia entre las ciudades.

Cuando tenemos un peso asociado a cada arista del grafo se dice que es un **grafo ponderado**.

2.2.4. Grado

Sea $G = (V, E)$ un grafo, se define como **grado** o **conectividad** de un nodo al número de aristas incidentes en dicho nodo. Otra forma de definir el grado de un nodo es a través de su vecindad, es decir, el grado de un nodo equivale al número de vecinos que tiene. A esta medida se le suele denotar por $gr(x)$, $grado(x)$ o $deg(x)$ en inglés y podemos definirla formalmente como:

$$deg(a) = \sum_{a \in V} w_{ij}$$

donde $w_{ij} = 1$ si la arista $\{i, j\} \in E$ es incidente o 0 si no lo es. Para los grafos ponderados, w_{ij} indica el peso de dicha arista. A los nodos con $deg(x) = 0$ se les llama **nodos aislados** y con $deg(x) = 1$ **nodos hoja**.

Existe una relación directa entre el grado de cada uno de los nodos y el número total de aristas del grafo, eso se conoce como **Lema del apretón de manos** y que definimos como:

$$2|E| = \sum_{a \in V} deg(a)$$

En caso de los grafos dirigidos, tendremos dos tipos de grados:

- De **entrada** en el que contarían solo las aristas que inciden de él.
- De **salida** en el que contarían solo las aristas que salen de él.

Otro elemento que debemos conocer es la **matriz de grados** ya que será usada en algunos algoritmos posteriormente. Se trata de una matriz diagonal:

$$D = \begin{bmatrix} \deg(1) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \deg(n) \end{bmatrix}$$

definida por:

$$D_{i,j} = \begin{cases} \deg(i), & \text{si } i = j \\ 0, & \text{en otro caso} \end{cases}$$

2.2.5. Representación Matricial

Una de las formas mas comunes de representación de grafos es mediante el uso de matrices debido a su simplicidad.

2.2.5.1. Matriz de adyacencia

Sea $G = (V, E)$ un grafo, definimos la **matriz de adyacencia** del grafo como una matriz $|V| \times |V|$ donde cada elemento se calcula como:

$$A_{i,j} = \begin{cases} 1, & \text{si } i \text{ y } j \text{ son adyacentes} \\ 0, & \text{en otro caso} \end{cases}$$

Para grafos ponderados, se cambiará el valor 1 por el peso de la arista que une a los nodos i y j .

2.2.5.2. Matriz de incidencia

Sea $G = (V, E)$ un grafo, definimos la **matriz de adyacencia** del grafo como una matriz $|V| \times |E|$ donde cada elemento se calcula como:

$$M_{i,j} = \begin{cases} 1, & \text{la arista } j \text{ es incidente al nodo } i \\ 0, & \text{en otro caso} \end{cases}$$

2.2.6. Subgrafos

Sea $G = (V, E)$ un grafo, un subgrafo de G es un grafo cuyo conjunto de nodos es un subconjunto de nodos de G y el conjunto de aristas es un subconjunto de E . Podemos definirlo más formalmente como $G' = (V', E')$ es un subgrafo de G si:

- $V' \subseteq V$
- $E' \subseteq E$
- $G' = (V', E')$ es un grafo

Usando la definición anterior, decimos que G' es un supergrafo de G si G es un subgrafo de G' .

Sea $G = (V, E)$ un grafo, decimos que G es un grafo completo si, para cada par de nodos, existe una arista que los une. Se denota como K_n donde n es el número de nodos. Como ejemplos, podemos ver en las imágenes de debajo una representación de K_3 y K_5 . Como observación, indicar que K_3 es un subgrafo de K_5 .

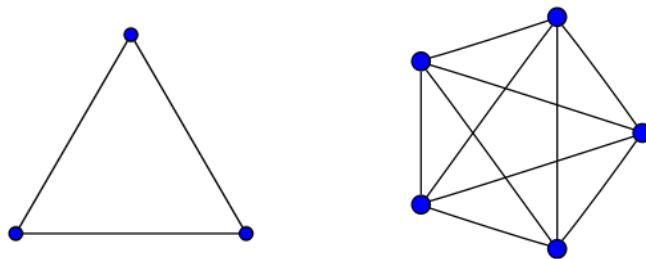


Figura 1. K_3 y K_5

Un **clique** en un grafo no dirigido G es un conjunto de nodos en el que existe una arista entre cada par de ellos, es decir, forman un **grafo completo**. El tamaño del clique se corresponde con el número de nodos que contiene. En el ejemplo anterior, el K_3 es un clique dentro de K_5 . Buscar un clique de un tamaño dado dentro de un grafo es un problema NP-Completo y se conoce como **Problema del Clique**. Una propiedad interesante es que los *cliques* pueden solaparse. Como nota decir que la palabra clique proviene del inglés y significa *conjunto de personas con intereses comunes*, definición muy acertada en el contexto que nos movemos.

Un **clique máximo** es un clique que no puede aumentarse añadiendo nodos adyacentes. Por ejemplo, si numeramos los nodos de nuestro K_5 del 1 al 5, el K_3 podría formarse seleccionando los nodos 1, 2 y 3. Este clique formado no es máximo, ya que añadiendo el nodo adyacente 4 y sus enlaces obtenemos K_4 .

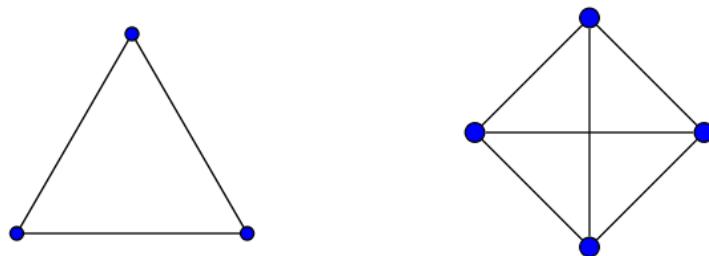


Figura 2. K_3 y K_4

Debido a las restricciones tan fuertes que se dan para que existan cliques dentro de los grafos y a que no se asemeja a las estructuras reales que nos rodean, se ha encontrado la necesidad de relajar algo esta definición. Por ello, se dan soluciones alternativas mediante estos elementos:

Un **k -clique** es un subconjunto maximal de nodos conectados entre si en el que la distancia máxima del camino entre cada par de ellos es k . En algunas ocasiones, este termino se refiere también al clique de tamaño k por lo que deberemos tener cuidado cuando lo veamos. Por ejemplo, un 2-clique se presenta mucho en redes sociales y tiene el significado de *amigos de amigos*.

Un **k -plex** es un subconjunto maximal con la propiedad de que cada nodo esta conectado a, como mínimo, $n - k$ nodos del subconjunto, donde n es el numero de nodos del subgrafo. Por lo tanto, si $k = 1$ se formará un clique ya que estarán unidos todos entre si, menos con el mismo. Al igual que los cliques, los k -plexes pueden solaparse.

Un **k -core** es un subconjunto maximal con la propiedad de que cada nodo esta conectado a, como mínimo, k nodos del subconjunto. Por lo tanto, podemos ver que un k -core es un $(n - k)$ -plex. Una propiedad interesante es que los k -core pueden contener k -cliques y k -plexes y a diferencia de estos últimos, los k -core no pueden solaparse ya que dos k -core que comparten uno o varios nodos formarían un k -core más grande.

Al igual que la detección de los cliques, la búsqueda de k -cliques y k -plexes se convierte en un problema NP-Completo por lo que los algoritmos de Detección de Comunidades basados en detección de estos grupos suelen ser muy costosos. Sin embargo, la detección de k -core puede resolverse en tiempo polinomial por lo que suelen usarse como etapa de pre-procesamiento en la detección de cliques, k -cliques y k -plexes. Un algoritmo sencillo de detección de k -core consiste en eliminar todos lo nodos de la red que tienen un grado menor a k de forma recursiva ya que, por definición, si un nodo tiene un grado menor a k no puede pertenecer a un k -core.

2.3. Comunidades

El término **Comunidad** se usa a menudo en numerosos contextos con diferentes connotaciones. Probablemente, la Sociología fue la primera en usar este término para de notar a un conjunto de personas con intereses y actividades comunes. Cuando las redes se establecieron como modelo a seguir para representar sistemas complejos, el concepto de comunidad se expandió para referirse a los grupos formados en estas redes con un alto grado de cohesión interna y con un alto grado de separación entre ellos donde los elementos no tienen porque ser personas.

Aunque no existe una única definición de comunidad, desde un punto de vista matemático y siendo $G = (V, E)$ un grafo, podemos definir a una comunidad como un subgrafo de G en el que sus nodos están más densamente conectados que con el resto. En la siguiente imagen podemos visualizar claramente tres comunidades donde se aprecian muy bien las definiciones anteriores de comunidad.

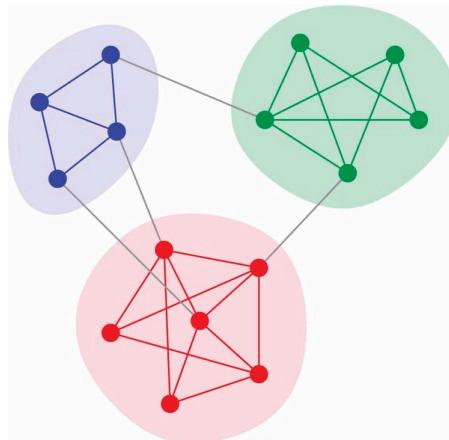


Figura 3. Ejemplo Comunidades Newman

Debemos resaltar las características de comunidad que definimos arriba como son alta cohesión interna de la comunidad y alta separación entre ellas (visibles en la imagen) ya que varios algoritmos de los que describiremos en el siguiente capítulo se apoyan en ellas para su detección.

2.3.1. Análisis local

Como hemos definido, una comunidad es un conjunto de nodos densamente conectados, hasta tal punto de que se podría analizar de manera independiente del resto de la red. El estudio de las comunidades que se basan en este concepto suelen centrar sus estudios en la búsqueda de cliques, k -cliques, k -plexes o k -core.

2.3.2. Análisis global

Otro enfoque de análisis, que es el que usaremos nosotros, plantea que no se puede analizar una comunidad por si sola ya que forma parte fundamental de la estructura comunitaria. La idea fundamental parte de la comparación de la red analizada con una generada de forma aleatoria, en la que no se espera que tenga una estructura de comunidad en su interior por lo que si se presentan diferencias entre ambas redes, se puede inferir que existe algún tipo de estructura comunitaria.

2.4. Medidas

Para la detección de comunidades es necesario resaltar las características propias de este tipo de estructuras, es decir, que características debe tener una agrupación de nodos para que se le pueda asociar el término *comunidad*. Para hacerlo de manera cuantitativa necesitamos definir una serie de medidas que nos ayudarán en el proceso de su detección.

2.4.1. Distancia

Sea $G = (V, E)$ un grafo, definimos la **distancia** entre dos nodos $n_i, n_j \in V$ como la longitud el camino más corto que los une, también llamado **camino geodésico**. Cabe destacar algunos casos especiales:

- Si $i = j$, $d(n_i, n_j) = 0$.
- Si n_i y n_j son adyacentes, $d(n_i, n_j) = 1$.
- Si no existe un camino entre n_i y n_j , $d(n_i, n_j) = \infty$.

En un grafo ponderado, la distancia se calcula como:

$$d(n_i, n_j) = \sum w_{ij}$$

donde w_{ij} indica el peso de cada una de las aristas que componen el camino.

Una de las medidas más importantes para conocer el tamaño de nuestra red es el **diámetro** como la máxima distancia que podemos encontrar entre cualquier par de nodos de la red. La definición formal sería:

$$\text{Diametro}(G) = \max\{ d(n_i, n_j) \} \forall n_i, n_j \in V.$$

2.4.2. Densidad

Sea $G = (V, E)$ un grafo, definimos la **densidad** de G como la relación que existe entre el número de aristas presentes en G y el número total de aristas que podrían existir. La definimos formalmente como:

$$\delta(G) = \frac{2m}{n(n - 1)}$$

donde $m = |E|$ y $n = |V|$. La densidad máxima para un grafo es 1, lo que conocemos como **grafo completo**.

Así mismo, podemos definir también la densidad a nivel de comunidad, tanto internamente como externamente. Sea $G' = (V', E')$ un subgrafo de G , la **densidad interna** es la relación entre el número total de aristas de G' y el número máximo posible de aristas de G' y se calcula como:

$$\delta_{interna}(G') = \frac{2m'}{n'(n'-1)}$$

donde $m' = |E'|$ y $n' = |V'|$.

Recíprocamente, definimos la **densidad externa** como la relación entre el número total de externas a G' y el número máximo posible de aristas externas a G' y se calcula como:

$$\delta_{externa}(G') = \frac{2(m - m')}{n'(n - n')}$$

2.4.3. Centralidad

Las medidas de **centralidad** sirven para detectar nodos que son relevantes o *céntricos* en una grafo. Conocer la centralidad de un elemento puede ayudar a determinar su impacto dentro del sistema, como por ejemplo una persona en una red social o la importancia de un servidor en internet. Esta medida no es un atributo natural de los elementos de una red, sino que es un atributo estructural, es decir, que depende estrictamente de su localización en la red.

2.4.3.1. Centralidad por Grado

Es la medida de centralidad más simple ya que la **centralidad por grado** de cada nodo equivale a su grado, que ya hemos descrito en apartados anteriores y que complementaremos en el siguiente párrafo.

Sea $G = (V, E)$ un grafo, $G' = (V', E')$ un subgrafo de G y un nodo $n \in V'$, definimos:

- **Grado interno** de n como el número de nodos, también pertenecientes a V' , a los que está conectado.
- **Grado externo** de n como el número de nodos, pertenecientes a V pero no a V' , a los que está conectado.

Teniéndose que:

$$C_{deg}(n) = deg(n)_{interno} + deg(n)_{externo}$$

Por lo tanto, el grado interno de un nodo nos permite saber la importancia del mismo dentro del subconjunto.

2.4.3.2. Centralidad por Cercanía

Sea $G = (V, E)$ un grafo, la **centralidad por cercanía** o *closeness centrality* se basa en calcular la suma o promedio de las distancias desde un nodo a todos los demás nodos de G . Formalmente, podemos definirla como:

$$C_{closeness}(i) = \frac{1}{\sum d_{ij}}$$

O en valor promedio:

$$C_{closeness}(i) = \frac{n}{\sum d_{ij}}$$

Esto implica que un nodo es más relevante mientras más cerca esté de los demás. Hay que tener en cuenta que esta medida no puede calcularse para nodos aislados.

Invirtiendo ambas ecuaciones tendríamos el valor recíproco conocido como **lejanía**, aunque no deja de ser la misma idea.

2.4.3.3. Centralidad por Vector Propio

Sea $G = (V, E)$ un grafo, la **centralidad por vector propio** o *eigenvector centrality* se basa en que la relevancia de un nodo en la red viene dada por los relevantes que sean sus nodos adyacentes, es decir, que los nodos que poseen un valor alto de centralidad están conectados a muchos nodos que a su vez están bien conectados. Esta medida se calcula como:

$$C_{eigenvector}(i) = \frac{1}{\alpha} \sum_j A_{ij} v_j$$

donde A es la matriz de adyacencia de G , v y α^{-1} son el vector propio y valor propio dominante respectivamente de la matriz de adyacencia.

Una variante muy relevante es el algoritmo de **PageRank** que diseño Google y es una pieza fundamental de su motor de búsquedas.

2.4.3.4. Centralidad por Intermediación

Sea $G = (V, E)$ un grafo, la **centralidad por intermediación** o *betweenness centrality* se basa en calcular la probabilidad que existe de que un nodo n_i se encuentre en un camino geodésico entre dos nodos n_j y n_k donde $n_i, n_j, n_k \in V$. De igual forma, esta misma idea puede aplicarse a las aristas. Esta medida fue propuesta por [Freeman \(1977\)](#). Lo que se pretende conseguir con esta medida es detectar nodos que sirvan de nexo entre comunidades o aristas las conecten entre sí, otorgándoles valores altos de centralidad de la siguiente forma:

$$C_{\text{betweenness}}(i) = \sum_{j,k} \frac{b_{jik}}{b_{jk}}$$

donde b_{jk} es el número total de caminos geodésicos entre los nodos n_j y n_k , y b_{jik} el número total de caminos geodésicos entre los nodos n_j y n_k que también pasan por n_i .

El problema que tiene esta medida es su complejidad ya que tenemos que calcular todos los posibles caminos mínimos entre cada par de nodos. Por ello, Newman (2005) propuso una alternativa que relajaba las restricciones establecidas basándose en que la elección del camino no siempre tenía porque ser la mejor en todos los casos. Su idea se centra en considerar caminos aleatorios y que además no tuviesen porque ser mínimos, de forma que se pudiese obtener un valor de centralidad aproximado más rápidamente.

2.4.4. Coeficiente de Agrupamiento

Sea $G = (V, E)$ un grafo, el **coeficiente de agrupamiento** o *clustering coefficient* se basa en determinar el agrupamiento de G detectando el número de triángulos en el interior del mismo, entendiendo como triángulos a los cliques de tamaño 3 ya mencionado en apartados anteriores como K_3 . Definimos formalmente al coeficiente de agrupamiento local como:

$$C_i = \frac{N_\Delta(i)}{N_3(i)}$$

donde N_Δ es el número de triángulos conectados al nodo i y N_3 es el número de caminos de longitud 2 que tienen a i como nodo intermedio.

Podemos calcular estas medidas de la siguiente forma:

$$N_{\Delta}(i) = \sum_{k>j} a_{ij} a_{ik} a_{jk}$$

y

$$N_3(i) = \sum_{k>j} a_{ij} a_{ik}$$

donde a_{ij} a_{ik} a_{jk} son elementos de la matriz de adyacencias.

Podemos reescribir la ecuación usando la misma idea que con la densidad, donde N_{Δ} sería el número aristas entre los vecinos de i y N_3 sería el número total de aristas que pudiera existir entre dichos vecinos, por lo que tendríamos la siguiente ecuación:

$$C_i = \frac{2m_i}{n_i(n_i - 1)}$$

donde m_i es el número de aristas entre los vecinos de i y n_i es el número de vecinos de i .

Para obtener el valor global de esta medida basta con calcular el promedio de los valores obtenidos para cada nodo de la definición anterior. Existe también otra definición de coeficiente de clustering global, más conocido como **coeficiente de transitividad** que se define como:

$$C = \frac{3N_{\Delta}}{N_3}$$

donde N_{Δ} es el número de triángulos totales de la red y N_3 es el número total de caminos de longitud 2 en la red. El factor $x3$ viene del hecho de que cada triángulo de la red puede formarse hasta tres veces, cada una con un vértice central diferente. Podemos calcular las demás medidas de la siguiente forma:

$$N_{\Delta} = \sum_{k>j>i} a_{ij} a_{ik} a_{jk}$$

$$N_3 = \sum_{k>j>i} a_{ij} a_{ik} + a_{ji} a_{jk} + a_{ki} a_{kj}$$

donde a_{ij} a_{ik} a_{jk} son elementos de la matriz de adyacencias.

Aunque ambas definiciones de coeficiente de agrupamiento global parecieran dar el mismo resultado, no es así ya que el coeficiente de transitividad da el mismo peso a cada triángulo de la red, mientras que el coeficiente de agrupamiento da el mismo peso a cada vértice.

2.4.5. Modularidad

Esta medida fue propuesta por [Newman et al. \(2004\)](#) para la detección de comunidades en una red no aleatoria. Sea $G = (V, E)$ un grafo, se basa en calcular densidades dentro y fuera de la comunidad y compararlas con la densidad general que tendría G si fuese aleatorio. Podemos definirla de manera formal como:

$$Q = \frac{1}{2m} \sum_{i,j} \left(a_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

donde m es el número de aristas del grafo, a_{ij} es un elemento de la matriz de adyacencia de G . $\frac{k_i k_j}{2m}$ representa el número de conexiones esperadas al azar teniendo en cuenta los grados k_i y k_j que son los grados de los nodos i y j . Por último, $\delta(c_i, c_j)$ vale 1 si nodos i y j pertenecen a la misma comunidad y 0 en caso contrario.

Esta medida está acotada entre $[-1/2, 1]$, siendo los valores entre $[0.3, 0.7]$ considerados como buenos si buscamos estructuras de comunidad en la red.

Podemos apreciar viendo la formula, que lo que trata de cuantificar es cómo de conectados están los nodos en cada clúster, teniendo en cuenta también el número de conexiones que tiene cada nodo.

Por ejemplo, un nodo (A) que sea **únicamente adyacente a todos los nodos de su clúster** aportará mejores coeficientes ($a_{ij} - \frac{k_i k_j}{2m}$) a la modularidad que otro (B) que está conectado con casi **todos los nodos de la red**. A pesar de que a_{ij} en ambos casos valdrá 1, $k_i k_j$ se disparará en el segundo caso; ya que se está penalizando el hecho de que el nodo no esté conectado exclusivamente con los nodos de su clúster.

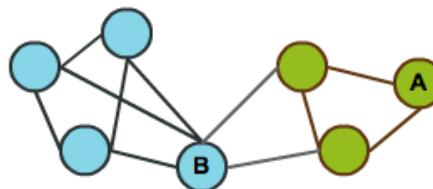


Figura 4. Ejemplo Modularidad

3. Detección de Comunidades

Detectar la estructura de comunidad en una red no es un problema actual, pero si que está en auge debido al crecimiento de los últimos años de las redes sociales ya que es la aplicación más directa que se nos podría ocurrir aunque, por supuesto, no es la única. Debido a su alta complejidad computacional, las ciencias de la computación también se hayan interesado por él, lo que ha permitido el nacimiento de varias familias de algoritmos que abordan y resuelven este problema. Estos algoritmos pertenecen a la rama de **Aprendizaje No Supervisado**, concretamente son Algoritmos de Agrupamiento o *Clustering*. En el siguiente capítulo se hará una revisión de los métodos más representativos clasificados en varias familias atendiendo a su forma de agrupar los nodos de la red.

Usaremos como parámetro de clasificación el enfoque que aplique el método a la resolución de nuestro problema, en nuestro caso tendremos el enfoque clásico particional, jerárquico, Greedy modular o espectral. Otra cuestión que debemos abordar es si permiten detectar comunidades con solapamiento o no, es decir, permiten que cada nodo pueda pertenecer a más de una comunidad a la vez o, por el contrario, solo puedan pertenecer a una única comunidad. Nosotros nos hemos centrado en los algoritmos sin solapamiento, que fueron los primeros que dieron solución al problema que se nos plantea, aunque los estudios y técnicas actuales se centran en permitir el solapamiento para mejorar las técnicas clásicas que aquí presentaremos.

3.1. Métodos Particionales

3.1.1. Particionamiento sobre un Espacio Métrico

Este conjunto de métodos recoge las técnicas clásicas de clustering basadas en agrupar un conjunto de puntos de un espacio métrico, de modo que cada nodo corresponde a un punto usando como medida de similitud cualquier medida de distancia en el espacio métrico como puede ser la distancia euclídea, distancia de Manhattan, etc. Estos métodos ya tienen pre-assignado el número de particiones a realizar y el objetivo fundamental es separar los puntos en k particiones minimizando (o maximizando) una función de coste que viene dada por la medida de distancia entre los puntos. En esta familia podremos agrupar a métodos como:

- **Minimum k -Clustering:** Se centra en minimizar el diámetro de los clusters de tal forma que se generen clusters compactos.

- **Min-Sum k -Clustering:** Se centra en minimizar la cohesión dentro de los clusters, es decir, en minimizar la distancia media entre cada par de nodos dentro de cada clúster.
- **k -Center:** Para cada clúster define un centroide y se encarga de minimizar la distancia máxima de este centroide a los demás puntos del clúster.
- **k -Means:** Tiene un funcionamiento idéntico al k -Center con la salvedad que reemplaza la distancia máxima por la distancia media.

Evidentemente, estos no son los únicos métodos de su clase, aunque quizás si los más representativos. Como se puede apreciar, el funcionamiento interno de estos algoritmos es muy similar por lo que nos centraremos en el algoritmo **k -Means** ya que es el más conocido y el más utilizado.

3.1.1.1. k -Means

El algoritmo de las k -Medias es considerado uno de los algoritmos de clustering más sencillos y más populares en la literatura propuesto por [MacQueen \(1967\)](#). Como ya indicábamos, se le asocia a cada partición un punto de referencia llamado centroide, el cual se considera centro geométrico del clúster y su objetivo es generar k particiones del conjunto de elementos en el que la distancia media de los elementos de un grupo a su centroide sea lo más pequeña posible.

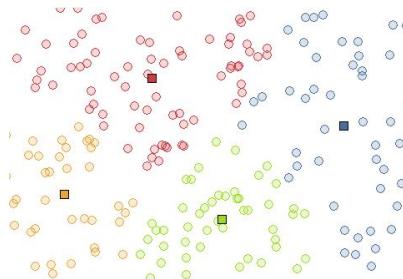


Figura 5. KMeans (Fuente: Univ. Carlos III de Madrid)

Para poder aplicar el algoritmo k -Medias genérico a nuestra red tenemos que transformarla en un conjunto de puntos de un espacio métrico puesto que no podemos asegurar que nuestros nodos contengan atributos mediante los cuales poder compararlos, para ello usaremos un algoritmo de diseño de grafos sobre un plano llamado **Algoritmo de Fruchterman-Reingold**, propuesto por [Fruchterman y Reingold \(1991\)](#), el cual se basa en fuerzas de atracción y repulsión similares a muelles y fuerzas eléctricas respectivamente entre los nodos para asignarles un valor x e y .

La función de coste a minimizar en este algoritmo se conoce como suma de errores cuadráticos y se calcula como:

$$SSE = \sum_{i=1}^k \sum_{n \in i} d(m_i, n)^2$$

donde d es la distancia euclídea entre un par de elementos usando sus atributos y se calcula como:

$$d(n_1, n_2) = \|n_1 - n_2\| = \sqrt{(n_{11} - n_{21})^2 + \dots + (n_{1l} - n_{2l})^2}$$

y m_i el centroide asociado al cluster i , que calcularemos como:

$$m_i = |C_i|^{-1} \sum_{n \in i} n$$

Una vez definidos los diferentes componentes del algoritmo (transformación al espacio métrico, función de coste, distancia entre elementos y cálculo del centroide), pasamos a su explicación paso a paso:

1. Realizamos un preprocessamiento en la red para asignar atributos a nuestros nodos. En nuestra implementación, por ejemplo, se aplicará el **Algoritmo de Fruchterman-Reingold** como preprocessamiento para asociar cada nodo con un punto del espacio. **Este paso no es necesario si ya partimos de una red con atributos en los nodos con los que poder compararlos.**
2. Elegimos, de forma aleatoria, los k centroides correspondientes a los clusters.
3. Mientras la disminución en el valor obtenido mediante función de coste en cada iteración supere un umbral μ , el cual usaremos como criterio de parada. También se suele establecer un número máximo de iteraciones a realizar.
4. Calculamos las distancias entre cada nodo y los centroides, asignando cada nodo al clúster del centroide más cercano.
5. Recalculamos los centroides a partir de los clusters generados en el paso anterior.
6. Repetimos los pasos 4 y 5 hasta que se cumpla el criterio de parada del paso 3.

Uno de los problemas que plantea este método se encuentra en el paso 4, ya que puede darse el caso de que haya clusters vacíos debido a que sus centroides no eran cercanos a ningún punto, por lo que se podría dar el caso de obtener menos particiones de las deseadas.

Otro inconveniente de estos algoritmos es que debes establecer el número de particiones al inicio, lo que los hace poco recomendables si no se hace un estudio previo sobre la red o una estimación de las comunidades de la misma.

La complejidad computacional de este algoritmo es $O(nkdi)$ donde n es el número de nodos, k es el número de clusters a generar, d es el número de dimensiones de nuestro espacio métrico e i es el número de iteraciones a realizar. Dado que para pasar de la red a un espacio métrico, aplicamos el algoritmo de distribución Fruchterman-Reingold, el cual es de orden cuadrático, el algoritmo K-medias que hemos implementado se ve acotado por esta eficiencia.

3.1.2. Particionamiento sobre grafos

El particionamiento de grafos consiste en dividir el grafo en k particiones minimizando una función objetivo, conocida como **función de corte**, que nos permitirá construir el conjunto de particiones en sucesivas iteraciones. Fijar el número de particiones es necesario ya que sin esta restricción tendríamos que la solución de menor corte sería añadir todos los nodos a la misma partición, donde el corte valdría 0, por lo que solo tendríamos un solo grupo. Existen diferentes enfoques de particionamiento que describiremos a continuación.

3.1.2.1. Corte Mínimo o Min Cut

El particionamiento por **corte mínimo** trata de minimizar el número de aristas que unen los diferentes grupos, es decir, minimizar la función de corte definida como:

$$Cut(C_1, C_2, \dots, C_k) = \frac{1}{2} \sum_{i=1}^k W(C_i, \bar{C}_i)$$

donde C_1, C_2, \dots, C_k es un posible conjunto de particiones, \bar{C}_i es un cluster diferente a C_i y W es la suma de las aristas que unen los dos clusters dados, que definimos como

$$W(C_r, C_t) = \sum_{i \in C_r, j \in C_t} a_{ij}$$

Si nuestro grafo es ponderado, cambiaremos a_{ij} por w_{ij} , elemento de la matriz de pesos de nuestro grafo.

El problema que plantea este enfoque de particionamiento es que podría ser necesario imponer un tamaño a las particiones. Si dejásemos este parámetro libre, podríamos encontrarnos que la solución de corte mínimo sería separar los $k - 1$ nodos de grado menor del resto de nodos. Por este motivo no es recomendable en redes donde existan nodos hoja, ya que sin esta restricción las soluciones se volverían triviales y generaría particiones poco razonables.

3.1.2.2. Corte Mínimo Normalizado o NCut

El particionamiento por **corte mínimo normalizado** propone normalizar la medida de corte mínimo usando el volumen de cada subconjunto:

$$NCut(C_1, C_2, \dots, C_k) = \frac{1}{2} \sum_{i=1}^k \frac{W(C_i, \bar{C}_i)}{vol(C_i)}$$

donde $vol(C_i)$ es el volumen del subconjunto C_i medido como la suma de los grados de los nodos pertenecientes a C_i :

$$vol(C_i) = \sum_{j \in C_i} deg_j$$

3.1.2.3. Mínimo Radio de Corte o RadioC

El particionamiento por **mínimo radio de corte** se plantea como alternativa para solventar el problema de nodos con bajo grado en la red dividiendo entre el tamaño del subgrupo:

$$RadioCut(C_1, C_2, \dots, C_k) = \frac{1}{2} \sum_{i=1}^k \frac{W(C_i, \bar{C}_i)}{|C_i|}$$

3.1.2.4. Corte Min-Max o Min-Max Cut

El particionamiento por **corte max-min** trata de minimizar la similitud entre los grupos maximizando la similitud interna de cada uno de ellos, es decir, teniendo en cuenta el número de aristas dentro del clúster frente al número de aristas que lo une a los demás. Lo definimos de manera más formal como:

$$MinMaxCut(C_1, C_2, \dots, C_k) = \frac{1}{2} \sum_{i=1}^k \frac{W(C_i, \bar{C}_i)}{W(C_i, C_i)}$$

3.1.2.5. Kernighan-Lin

Uno de los usos más comunes para el particionamiento de grafos es el conocido como **Bisección mínima**, donde el número de particiones se fija a 2, muy usado en diseño de circuitos electrónicos. El algoritmo de **Kernighan-Lin**, propuesto por [Kernighan et al. \(1970\)](#), es uno de los algoritmos clásicos de bisección de grafos propuesto para dar solución al problema de diseño de circuitos, minimizando el número de conexiones entre los módulos facilitando su diseño y ensamblaje. Este algoritmo sigue una heurística de particionamiento balanceado de forma iterativa, intercambiando elementos entre los dos subconjuntos de forma que se realice un corte mínimo entre ambos. La elección de pareja de nodos a intercambiar se realiza usando una función de coste asociado al intercambio, el cual se usará para calcular la función de ganancia que debemos maximizar en cada iteración.

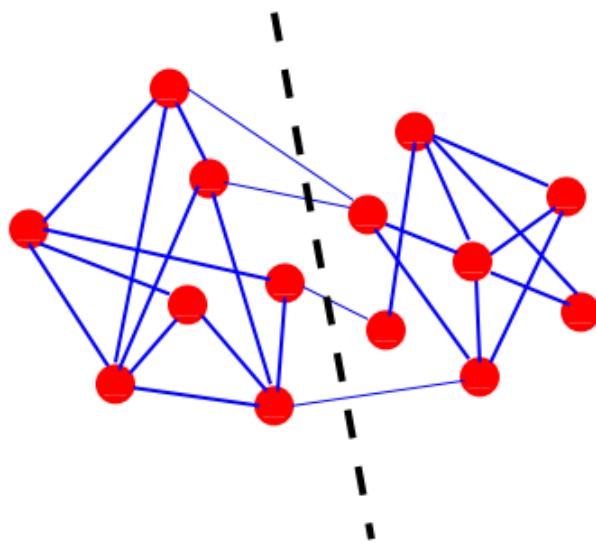


Figura 6. Kernighan-Lin

Sea $G = (V, E)$ un grafo, en el algoritmo se indica que la bisección debe separar G en dos subconjuntos A y B donde $A \cup B = V$, $A \cap B = \emptyset$ y $|A| = |B| = \frac{|V|}{2}$. Esta última restricción debe relajarse para poder permitir un conjunto de nodos impar. Este algoritmo funciona tanto en grafos ponderados como en grafos sin ponderar. Para estos últimos, tendremos que cambiar el peso de la arista por el elemento de la matriz de adyacencia correspondiente.

Como hemos indicado, necesitamos calcular el coste del intercambio de los nodos puesto que nos permitirá elegir qué parejas debemos seleccionar. Para ello definiremos dos tipos de costes asociados a cada nodo. El **coste interno** de un nodo se define como la suma de los pesos de las aristas que lo unen a los demás nodos de su clúster, es decir:

$$I_i = \sum_{j \in A} w_{ij}$$

donde A es el conjunto de nodos al que pertenece el nodo i .

De manera similar, el **coste externo** de un nodo se define como la suma de los pesos de las aristas que lo unen a los nodos del otro clúster, es decir:

$$E_i = \sum_{j \in B} w_{ij}$$

donde B es el conjunto de nodos al que no pertenece el nodo i .

Usando los dos elementos anteriores, definimos el **coste total** de un nodo como la diferencia entre su coste externo y su coste interno. De manera formal sería:

$$D_i = E_i - I_i$$

La función de **ganancia** que debemos maximizar se define como el coste de intercambio de dos nodos i y j donde $i \in A$ y $j \in B$ y se calcula como:

$$g(i, j) = D_i + D_j - 2w_{ij}$$

Otra cosa a tener en cuenta en las sucesivas iteraciones del algoritmo es establecer una forma de no intercambiar nodos que ya hayan sido intercambiados, situación que puede darse debido a que siempre se intercambiarán nodos, aunque la ganancia sea negativa, siempre y cuando sea la mejor opción. Para ello bloquearemos la pareja de nodos de tal forma que no puedan volver a ser intercambiados en la misma iteración, hecho que usaremos como criterio de parada en cada ejecución, cuando no haya más nodos para intercambiar.

Dicho esto, el algoritmo se resume de la siguiente forma:

1. Asignamos, de forma aleatoria o mediante alguna heurística, cada uno de los nodos a uno de los dos subconjuntos A y B de tal forma que se cumplan las restricciones impuestas arriba.

2. Seleccionamos la pareja de nodos, de entre los que no estén bloqueados, cuyo intercambio produzca una mayor ganancia. Si no hay nodos disponibles, saltamos al paso 4.
3. Intercambiamos ambos nodos y los bloqueamos, volviendo al paso 2.
4. Calculamos la ganancia máxima producida sumando, en orden, las diferentes ganancias producidas por los intercambio, seleccionando las k primeras parejas que produzcan esa ganancia.
5. Si la ganancia es positiva, realizamos los k intercambios en los conjuntos originales, desbloqueamos todos los nodos y volvemos al paso 2. Si la ganancia máxima producida por los intercambios es negativa, es decir, ninguno de los intercambios ha tenido algún efecto positivo, terminamos.

La complejidad computacional de este algoritmo es $O(n^3)$ por iteración, aunque en implementaciones muy eficientes se podría alcanzar $O(n^2 \log n)$. Lo que ocurre es que debido a que el número de iteraciones suele ser bastante bajo, no suele tenerse en cuenta o se suele establecer como una constante. En el peor de los casos podríamos estar hablando de un algoritmo de orden por $O(n^4)$.

3.2. Método Jerárquicos

La base de estos algoritmos es determinar una jerarquía con distintos niveles de agrupamiento que típicamente suelen ser representados mediante un *dendograma*. Así, cada nodo sería una hoja del dendrograma que aparecería en el nivel más bajo formando un grupo aislado y conforme vamos ascendiendo se irían uniendo formando grupos cada vez más poblados hasta llegar al nivel más alto donde estarían la totalidad de nodos que conforman la red.

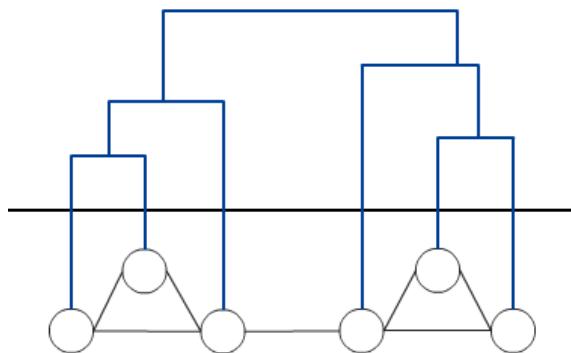


Figura 7. Dendrograma

En ocasiones no es necesario construir el dendograma completo y basta con regirse por algún criterio de parada, por ejemplo, determinar un cierto número de grupos o establecer un tiempo máximo para la ejecución del algoritmo.

Una vez conseguida esta clasificación jerárquica, la forma de elegir la distribución mas correcta puede estar influencia por diversos criterios: homogeneidad de los grupos, número predeterminado clusters, o, como en nuestro caso, aquella distribución que optimice alguna medida de calidad, por ejemplo la modularidad.

La clave de este tipo de métodos es tener algún tipo de medida de similitud entre elementos para poder realizar comparaciones y elegir en cada momento la pareja a tratar.

Orientado a la detección de comunidades en redes complejas surgen dos formas de realizar el agrupamiento:

1. **Métodos divisivos:** parten de un único grupo global que contiene a todos los nodos y los va separando formando subgrupos mediante la eliminación de enlaces. La complejidad de esta técnica recae en la forma de elegir qué enlace eliminar en cada iteración, con el objetivo de ir desligando nodos con baja interacción entre ellos.
2. **Métodos aglomerativos:** cada nodo empieza siendo un clúster aislado, los cuales se van juntando añadiendo enlaces entre ellos. De nuevo, la elección del enlace a añadir para unir dos clusters vuelve a ser la clave de la técnica, buscando ahora unir nodos que interactúen fuertemente.

3.2.1. Métodos divisivos

Estos métodos son llamados también métodos top-down por el hecho de ir desde "arriba" (el grupo global) hacia "abajo" (grupos de un solo elemento). El grupo global se denomina *clúster raíz* y este se va dividiendo iterativamente en *clusters* cada vez más pequeños hasta llegar a conjuntos de un solo nodo.

3.2.1.1. Método de Newman y Girvan

La labor de investigación de [Newman y Girvan \(2004\)](#) dio paso al estudio de la detección de comunidades aplicado a la Física. Su pilar es la medida que utilizaron para determinar los enlaces a eliminar durante el proceso de ejecución. Se conoce como *betweenness* y viene a ser la forma de cuantificar la intermediación de un enlace, esto es, la cantidad de caminos en los que se ve incluido el enlace con respecto a la totalidad de caminos mínimos que existen en una red para conectar todos los nodos entre sí.



Figura 8. Link Betweenness

Básicamente el algoritmo se rige por estos 3 pasos:

1. Medir el *betweenness* de todos los enlaces de la red.
2. Determinar el enlace con mayor medida.
3. Eliminar dicho enlace.
4. Volver a aplicar (retornar al paso 1) mientras queden enlaces sin quitar.

De esta forma en cada iteración, estamos borrando el enlace con mayor tráfico. Un enlace será muy visitado si une muchos nodos que no pueden conectarse por otro camino más corto, lo cual lleva a pensar que pertenecen a grupos distintos ya que los nodos de un mismo grupo están unidos por enlaces directos donde el *betweenness* es bajo.

El algoritmo termina cuando todos los enlaces hayan sido borrados, dejando como resultado un dendograma según los grupos que se fueron formando durante su ejecución.

Basándonos en la experiencia, podemos decir que los resultados que proporciona son bastante buenos con respecto al resto de técnicas aunque en cuanto a complejidad computacional y por lo tanto tiempo de ejecución, este algoritmo se ve limitado. La eficiencia radica principalmente en el cálculo de la medida *betweenness*, la cual está en el orden de $O(mn(m + n))$ y puede llegar a ser $O(n^3)$ cuando m se acerque mucho a n , siendo m el número de enlaces y n el de nodos.

3.2.1.2. *Método de Radicchi*

[Radicchi et al. \(2004\)](#) propusieron esta técnica que parte de la siguiente idea: una comunidad contiene nodos fuertemente interconectados donde se dan muchos bucles de nodos y enlaces, sin embargo los enlaces que conectan unas comunidades con otras se ven menos involucradas en bucles. Para entenderlo nos apoyamos en la siguiente figura.

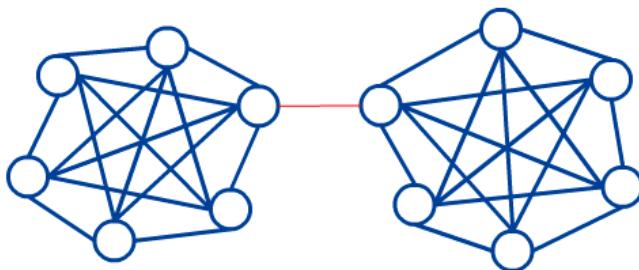


Figura 9. Coeficiente de agrupamiento de enlaces

Partiendo del concepto anterior, basta con determinar una medida que sea capaz de cuantificarlo y a partir de ahí, aplicar un método de detección de comunidades jerárquico divisivo. Por lo que nos centraremos en esta ocasión en el **coeficiente de agrupamiento de enlaces**.

La definición de esta medida se basa en el principio del **coeficiente de agrupamiento de nodos**, trabajo desarrollado por Duncan Watts y Steven Strogatz hace relativamente poco (1998). Consiste en determinar la proporción entre el numero de triángulos en los que se ve envuelto el nodo, con respecto al total que pudieran formarse.

Si esta idea la extrapolamos a aristas, tendríamos que tener en cuenta a los dos nodos que están conectados por dicho enlace. Así, dada la siguiente fórmula, podríamos calcular el coeficiente de agrupamiento de cada arista:

$$C_{ij} = \frac{z_{ij} + 1}{\min(k_i - 1, k_j - 1)}$$

- z_{ij} es el número de triángulos donde el nodo i y j aparecen
- k_i es el grado del nodo i

La ecuación esta planteada para tener controlar aristas que no estén en ningún triangulo gracias al +1 del numerador. En el denominador no debería darse nunca el valor 0 si entre los 2 nodos existe realmente una arista.

Los valores que se obtiene pueden considerarse inversos a los que nos ofrece la medida *betweenness* del algoritmo de Newman y Girvan. Cuando el *betweenness* crece, el coeficiente de agrupamiento disminuye y viceversa.

En términos de eficiencia, este algoritmo se comporta mejor ya que es de orden $O(n^2)$ pero en cuanto a resultados, aunque buenos, no llega a la calidad del método de Newman y Girvan. Cabe destacar que durante la ejecución, cuando se borre un enlace, el recálculo de la medida no tiene que hacerse en todas las aristas; bastará con hacerlo en aquellas que conecten con alguno de los nodos de la arista que se acaba de borrar.

El algoritmo tiende a fallar cuando el coeficiente promedio de agrupamiento de la red es bajo, ya que todas las aristas presentan valores pequeños y parecidos. Pero, por otro lado, funciona muy bien en redes donde existe un gran número de triángulos, como en caso de las redes sociales.

3.2.2. Métodos aglomerativos

Los métodos aglomerativos, también denominados *bottom-up*, funcionan de forma inversa a los divisivos como ya conocemos. Cada nodo forma un único clúster inicialmente, que tras el proceso de fusión de pares se van formando clusters cada vez más grandes hasta llegar a un único clúster global en el que se encuentran todos los nodos.

Para decidir que pares fusionar en cada iteración se plantean varias formas de determinar la distancia entre nodos: *Simple-Link*, *Complete-Link* y *Average-Link*.

El procedimiento a seguir vendría definido por los siguientes pasos:

- Inicializar cada nodo en un clúster diferente.
- Fusionar los clusters mas cercanos (según la medida de distancia)
- Volver a aplicar (retornar al paso 2) mientras queden clusters separados.

3.2.2.1. *Simple-Link*

Sibson (1973) propone un método eficiente de conexión simple que trata de encontrar la pareja de clusters más cercanos entre sí encontrando la pareja de nodos cuya distancia sea mínima. Teniendo en cuenta que la distancia entre dos nodos, definida en el apartado 2.4.1, la distancia entre dos clusters viene definida por:

$$d(C_i, C_j) = \min d(x, x'), x \in C_i, x' \in C_j$$

- $d(x, x')$ sería la distancia entre nodos y se evaluaría para cada par de nodos siendo uno de un clúster y otro del otro clúster.

La eficiencia algorítmica del algoritmo queda acotada por $O(n^3)$.

Con esta medida surge el problema del *encadenamiento* ya que dado un clúster siempre vamos a encontrar un clúster vecino a distancia 1, lo cual nos llevaría a unir ambos clusters sin tener en cuenta a todos los nodos que los componen, sino solamente a la pareja de nodos más cercanos.

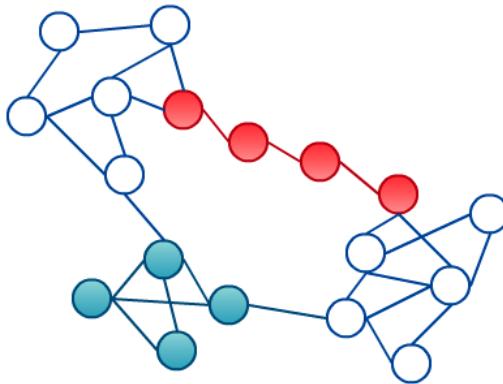


Figura 10. Problema del encadenamiento

3.2.2.2. *Complete-Link*

[Defays \(1977\)](#) propone un método eficiente de conexión completa que trata de encontrar la pareja de clusters más cercanos entre sí encontrando la pareja de nodos cuya distancia sea máxima. Se basa por tanto en la medida de distancia entre clusters, la cual queda definida en la siguiente fórmula:

$$d(C_i, C_j) = \max d(x, x'), x \in C_i, x' \in C_j$$

La eficiencia algorítmica de este método queda acotada por $O(n^3)$.

Su funcionamiento es mejor que el de *simple-link* ya que determina grupos más compactos y trata mejor el problema del encadenamiento, pero cuando se presentan nodos *outliers*, la medida de la distancia máxima entre clusters se ve afectada negativamente, siendo este, el talón de Aquiles del método *complete-link*.

3.2.2.3. *Average-Link*

El método de conexión media (UMPGA) surge para cubrir las deficiencias de los dos métodos anteriores, teniendo en cuenta a todos los nodos de los clusters para tomar la medida de distancia entre ellos. Las primera definición de este algoritmo se atribuye a [Sokal \(1958\)](#).

Así, en este caso la medida de distancia será una **media** de todas las distancias (a pares) entre los nodos de los dos grupos.

$$d(C_i, C_j) = \frac{\sum_{x \in C_i, x' \in C_j} d(x, x')}{|C_i| \cdot |C_j|}$$

De nuevo, la eficiencia algorítmica es $O(n^3)$ y sus resultados bastante mejores como veremos en la sección de pruebas.

3.3. Métodos Greedy basados en modularidad

Esta familia de algoritmos se basan en la medida de **modularidad** para evaluar la calidad de las divisiones en la red. Lo que pretenden es buscar agrupamientos que optimicen el valor de modularidad de la red, en un tiempo razonable ya que el cálculo de esta medida es cada vez más complejo conforme el tamaño de la red crece.

3.3.1. Método Fast Greedy

El procedimiento que sigue esta técnica es el que se propone en los métodos jerárquicos: se van añadiendo enlaces que formar iterativamente las agrupaciones.

La elección de la arista a añadir viene influenciada por la medida de la modularidad (ver capítulo 2.4 Medidas), entonces, el objetivo es juntar clusters e ir optimizando la modularidad general de la red hasta parar cuando no sea posible mejorar más.

La complejidad de esta técnica está acotada por $O(n^2m)$.

El cálculo de la modularidad es una tarea compleja que conlleva un coste computacional elevado por lo que las técnicas basadas en este método utilizan diferentes heurísticas con el fin de establecer un compromiso entre velocidad y calidad de los resultados. Incluso en algunas implementaciones se llega a trabajar con valores aproximados de modularidad para no emplear demasiado tiempo en esta medida. En nuestro caso hemos diseñado un procedimiento que nos da resultados considerablemente buenos en un tiempo razonablemente pequeño en comparación con otros métodos. Su funcionamiento se explica a continuación:

1. Inicialización
 - a. Separar todos los nodos en clusters diferentes.
 - b. Formar pequeños grupos con algún método particional sencillo como puede ser el *KMeans*. El número de grupos puede variar, por ejemplo, tantos grupos como la mitad de nodos es una buena inicialización en base a las pruebas que hemos realizado.
2. Optimización
 - a. Determinar un enlace (de forma aleatoria) que mejore la modularidad de la red cuando se añada.
 - b. Añadir el enlace.
 - c. Volver al paso 2.a (mientras queden enlaces que mejoren)

3.3.2. Multi Step Greedy

Este método fue propuesto por [Schuetz y Caflisch \(2008\)](#). Plantean un algoritmo Greedy *multi-pasos* combinado con un procedimiento de refinamiento local.

La técnica consiste de nuevo en juntar clusters en base a la mejora de modularidad que aporte el enlace a añadir, pero ahora, se escogerá el enlace que más optimice en cada momento. Además, como estrategia para ahorrar en cálculos de modularidad, podremos añadir varios enlaces si estos no están en contacto (no tienen nodos en común).

Utiliza una estructura denominada *QMatrix* donde se guardan las mejoras de modularidad que introduce cada enlace al ser añadido a la red. Su funcionamiento es el siguiente:

1. Inicialmente cada nodo es un clúster independiente.
2. Calcular *QMatrix*, esto es:
 - a. Para cada enlace que quede sin añadir a la red, comprobar cuánto cambiaría la modularidad si sólo dicho enlace fuese añadido a la red.
 - b. Si hay mejora, introducimos en la matriz una entrada *enlace-mejora*.
3. Recorrer *QMatrix* e ir añadiendo enlaces a la red:
 - a. Ordenar *QMatrix*: mejora descendente, enlace ascendente.
 - b. Por cada enlace, si el enlace no toca a ningún nodo de los que tocan los enlaces que ya han sido añadidos durante este paso, **añadir el enlace a la red**.
4. Volver al paso 2 (mientras queden enlaces en la red y se haya añadido algún enlace en la iteración anterior)

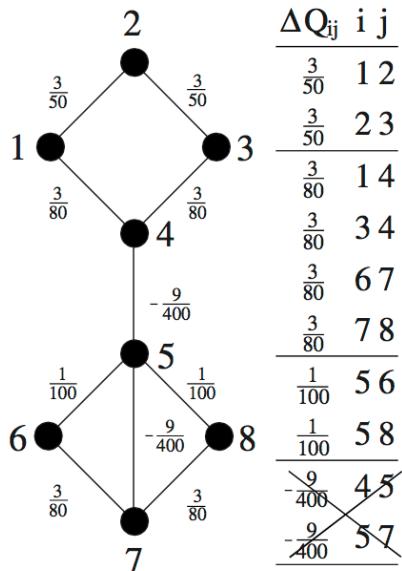


Ilustración 2. QMatrix (PHYSICAL REVIEW E 77, 046112 2008)

En la investigación de Philipp y Amedeo se propone también que el número de enlaces a añadir en cada iteración este limitado aunque en nuestros experimentos con redes de *Facebook* y *Twitter* los resultados no variaban mucho en calidad para el incremento de tiempo que suponía.

Determinar *QMatrix* tiene un coste computacional máximo de $O(mn^2)$, proceso que se repetirá i veces hasta que converja el algoritmo.

3.4. Métodos Espectrales

Los métodos espectrales pretenden dar solución al problema de particionamiento de grafos usando la descomposición en autovectores de la matriz de laplaciana de dicho grafo. Debido a su gran peso, se van a considerar como una familia independiente. [Von Luxburg \(2007\)](#) describe su funcionamiento general en dos pasos:

1. Transformar el conjunto de nodos en un conjunto del espacio métrico, cuyas coordenadas se corresponderán a los k vectores propios más relevantes de la matriz laplaciana.
2. Agrupar dichos puntos mediante alguna técnica de particionamiento en espacio métrico.

La principal ventaja que ofrecen estos algoritmos es que en el cambio de representación que introducen los autovectores se intensifican las características buscadas en una estructura comunitaria.

Hay que tener en cuenta que este campo de estudio está en pleno auge y no hay aún una teoría cerrada que explique completamente su funcionamiento y sus limitaciones por lo que nos centraremos en exponer las ideas y conceptos básicos sobre este tema y mostraremos algunos algoritmos de ejemplo.

3.4.1. Laplaciano de un grafo

Sea $G = (V, E)$ un grafo, una de las herramientas clave del clustering espectral es la **matriz laplaciana** asociada a G así como sus autovalores y autovectores. Dicha matriz se calcula como:

$$L = D - A$$

donde D es la matriz de grados asociada a G y A es la matriz de adyacencias de G . La matriz laplaciana tiene multitud de utilidades, aunque para nosotros la más importante es que representa la conectividad del grafo. Una propiedad interesante es que el número de autovalores iguales a 0 de L coincide con el número de grupos aislados en la red. El problema es que, por norma general, los diferentes grupos estarán unidos entre sí creando un grafo conexo, donde el cálculo de autovalores obtendría un único autovalor igual a 0 y los demás serán positivos. Sin embargo, si tenemos k grupos bien definidos en nuestra red, los k primeros autovalores serán cercanos a 0 permitiendo que los autovectores asociados puedan diferenciar claramente los grupos en un espacio de k dimensiones debido a que los componentes de estos vectores propios se parecerán entre sí formando grupos en un espacio métrico, que serán fácilmente agrupados mediante cualquier técnica de agrupamiento como *K-Means*.

Cabe destacar que dicha matriz no está normalizada. A su vez existen dos versiones normalizadas de la matriz cuya diferencia se centra en el proceso de normalización que siguen. Por una parte tenemos la **normalización simétrica** que definimos formalmente como:

$$L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$$

y la **normalización por caminos aleatorios** o *normalización asimétrica* que definimos formalmente como:

$$L_{rw} = D^{-1} L$$

3.4.2. Estimación del número de clusters

Una cuestión muy interesante que se nos plantea en el clustering espectral es la estimación del número de grupos que existe en nuestra red. Esto es posible gracias al análisis del espectro de la matriz laplaciana. Como hemos comentado en el apartado anterior, si tenemos k particiones bien definidas, los primeros k autovalores de la matriz laplaciana serán cercanos a 0, por lo que es de esperar que el autovalor $k + 1$ difiera bastante del autovalor k . Siguiendo esta idea, [Bielo y Hamad](#) propusieron el siguiente algoritmo como método de estimación del número de clusters.

Sea $G = (V, E)$ un grafo y L su matriz laplaciana, podemos estimar el número de grupos que existen en la red de la siguiente forma:

1. Obtenemos el conjunto de autovalores $\lambda_1, \lambda_2, \dots, \lambda_n$ asociados a L .
2. Ordenando el conjunto de autovalores crecientemente de tal forma que $0 \leq \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$.
3. Calculamos la distancia entre cada par de autovalores de forma creciente, es decir, $\lambda_{i+1} - \lambda_i$ para $i = 1..n - 1$.
4. Si la distancia máxima encontrada se encuentra en el par λ_j y λ_{j+1} , el número de clusters se corresponderá con j .

Hay que tener en cuenta que este es un método de estimación, por lo que no tenemos una garantía total de que el número de clusters estimado sea el número de clusters óptimo para la red. De hecho, para que el método funcione medianamente aceptable, la red deberá tener los clusters bien definidos.

3.4.3. Algoritmo genérico de particionamiento espectral

El siguiente algoritmo describe el método general en el que se inspiran los algoritmos específicos de clustering espectral para el particionamiento en k grupos:

1. Calculamos la matriz laplaciana L de nuestra red (normalizada o no).
2. Realizamos el cálculo de los autovectores de L .
3. Formamos una matriz U con los k autovectores propios más representativos de L en forma de columnas.
4. Interpretamos cada fila de U como un elemento del espacio métrico.
5. Agrupamos U mediante cualquier técnica de particionamiento en el espacio métrico.

La complejidad de este tipo de algoritmos está muy ligada la descomposición en autovalores y autovectores de la matriz laplaciana por lo que variará en función del algoritmo utilizado. Por norma general, suelen tener una complejidad $O(n^3)$.

Los siguientes algoritmos que presentaremos se basan en este añadiendo pequeñas modificaciones que los hacen únicos.

3.4.4. Bisección Espectral o EIG1

El algoritmo **EIG1** es un algoritmo espectral de bisección mínima de grafos propuesto por [Hagen y Kahng \(1992\)](#). Se basa en el cálculo del autovector correspondiente al segundo autovalor más pequeño de la matriz laplaciana. Este autovector se conoce como **autovector de Fiedler** cuyos elementos nos ayudarán en nuestra partición. Sea $G = (V, E)$ un grafo:

1. Calculamos la matriz laplaciana L de G .
2. Calculamos la descomposición en autovectores de L .
3. Obtenemos el autovector de Fiedler F .
4. Para cada valor $x_i \in F$ y su correspondiente nodo $n_i \in V$, si $x_i > \sigma$ asociamos el nodo n_i al cluster C_1 , sino asociamos el nodo n_i al cluster C_2 .

Se proponen diferentes métodos en la literatura para calcular el umbral σ , a partir del cual los nodos se asignarán a un cluster o a otro. El más sencillo se obtiene asignando a σ el valor 0, por lo que los elementos negativos se asociarán a un clúster y los positivos a otro. Otro umbral utilizado es asignar a σ la media del vector F , también se puede establecer como la mediana de F . Usando la misma idea que para la estimación de clusters, podemos establecer el umbral como el elemento de F cuya distancia con el siguiente, ordenados de forma creciente, sea máxima, de tal forma que si la distancia entre x_i y x_{i+1} es máxima, $\sigma = x_{i+1}$.

Como indicábamos en el apartado anterior, la complejidad de este algoritmo es cúbica, aunque puede reducirse usando una aproximación del algoritmo de Lanczos para el cálculo de un solo vector propio.

3.4.5. UKMeans

El algoritmo **UKMeans** es el algoritmo básico de particionamiento espectral propuesto por [Von Luxburg \(2007\)](#). Se inspira en el enfoque de corte mínimo usando las información que proporciona la matriz laplaciana sobre la conectividad del grafo. Sea $G = (V, E)$ un grafo:

1. Calculamos la matriz laplaciana L de G sin normalizar.
2. Calculamos los k vectores propio de L mas relevantes.
3. Generamos la matriz U estableciendo, como columnas, los vectores generados.
4. Interpretamos cada fila de U como un elemento del espacio métrico, ya que tendremos tantas filas como nodos en la red.

5. Aplicamos el algoritmo K-Means sobre U para generar las k particiones deseada.
6. Para cada el nodo $n_i \in V$, le asignamos el mismo clusters asignado al elemento $x_i \in U$.

Este algoritmo tiene un orden de eficiencia $O(n^3)$ (paso 2).

3.4.6. Algoritmo NJW o KNSC1

El algoritmo **NJW**, también conocido como **KNSC1**, es un algoritmo espectral reciente para resolver el problema del particionamiento de grafos desarrollado por [Ng et al. \(2002\)](#). Sigue un enfoque de corte mínimo normalizado por lo que usará un proceso de normalización simétrica para la matriz laplaciana. Sea $G = (V, E)$ un grafo:

1. Calculamos la matriz laplaciana L de G .
2. Realizamos la normalización simétrica de L obteniendo L_{sym} .
3. Calculamos los k vectores propio de L_{sym} mas relevantes.
4. Generamos la matriz U estableciendo, como columnas, los vectores generados.
5. Realizamos un nuevo proceso de normalización de U para obtener U' . Calcularemos cada valor $u'_{ij} \in U'$ como:

$$u'_{ij} = \frac{u_{ij}}{\sqrt{\sum_{f=1}^k u_{if}^2}}$$

6. Interpretamos cada fila de U' como un elemento del espacio métrico
7. Aplicamos el algoritmo K-Means sobre U' para generar las k particiones deseada.
8. Para cada el nodo $n_i \in V$, le asignamos el mismo clusters asignado al elemento $x_i \in U'$.

La complejidad de este algoritmo viene dada por el proceso de normalización de la matriz laplaciana. Puesto que este proceso requiere de multiplicación de matrices, además del propio cálculo de los vectores propios, el orden de eficiencia es $O(n^3)$.

3.5. Evaluación de los resultados

Tan importante como la detección de comunidades es evaluar cómo de buenas son las soluciones que obtenemos. Las medidas de evaluación se centran en resaltar una o varias características propias de una comunidad que ya mencionamos anteriormente. Como se debe suponer que no tenemos información de la red, aparte de la que nos proporciona la red en sí, presentaremos una serie de medidas, pertenecientes al conjunto de medidas de evaluación no supervisada, que nos ayudarán en el proceso.

3.5.1. Modularidad

Es la medida por excelencia para la evaluación de los resultados, hasta el punto de que existe una familia completa de algoritmos, ya mencionada, que la usan como función objetivo a optimizar. Ya se definió esta medida, propuesta por [Newman y Girvan \(2004\)](#), en el apartado 2.4.5, como:

$$Q = \frac{1}{m} \sum_{i,j} (a_{ij} - \frac{k_i k_j}{m}) \delta(c_i, c_j)$$

[Fortunato y Barthélemy \(2007\)](#) demostraron matemáticamente que la optimización de esta medida sufre de un límite de resolución. Esto significa que es incapaz de detectar comunidades de tamaño menor que un límite que viene determinado por el número de enlaces de las comunidades y el número de total de enlaces de la red.

3.5.2. Coeficiente de Cohesión

Una de las características de comunidad es que los nodos pertenecientes a la misma deben guardar una cierta semejanza entre ellos. Así, se define el **coeficiente de cohesión** como la suma de distancias entre un nodo y los demás nodos pertenecientes al mismo clúster. La cohesión de cada nodo se mide:

$$\text{Cohesión}(i) = \sum_{i,j \in C} \text{dist}(i, j)$$

También podemos calcular esta medida para cualquier clúster:

$$\text{Cohesión}(C) = \frac{1}{|C|} \sum_{i \in C} \text{Cohesión}(i)$$

O para toda la red completa:

$$\text{Cohesión}(\pi) = \frac{1}{|\pi|} \sum_{C \in \pi} \text{Cohesión}(C)$$

Siendo $\pi = C_1, C_2, \dots, C_k$ el conjunto de particiones establecido por el algoritmo. Por la naturaleza de esta medida se espera que sea lo más pequeña posible. La cohesión se acota en el intervalo $[0, \infty]$. El valor 0 se obtiene asignando todos los nodos a un clúster diferente.

3.5.3. Coeficiente de Separación

Otra de las características de comunidad es que los nodos pertenecientes a distintas comunidades deben ser lo menos parecidos entre sí. Así, se define el **coeficiente de separación** como la suma de distancias entre un nodo y los demás nodos pertenecientes a los clusters diferentes. La separación de cada nodo se mide:

$$\text{Separación}(i) = \sum_{j \in \bar{C}} \text{dist}(i, j)$$

Donde \bar{C} es todos los clusters que no son C . También podemos calcular esta medida para cualquier clúster:

$$\text{Separación}(C) = \frac{1}{|\bar{C}|} \sum_{i \in C} \text{Separación}(i)$$

O para toda la red completa:

$$\text{Separación}(\pi) = \frac{1}{|\pi|} \sum_{C \in \pi} \text{Separación}(C)$$

Donde $\pi = C_1, C_2, \dots, C_k$ el conjunto de particiones establecido por el algoritmo. Por la naturaleza de esta medida se espera que sea lo más grande posible. La separación se acota en el intervalo $[0, \infty]$. El valor máximo se obtiene asignando todos los nodos a un cluster diferente.

3.5.4. Coeficiente de Silueta

Una medida muy usada es el **coeficiente de silueta**, el cual mezcla los conceptos de cohesión y separación. [Rousseeuw \(1987\)](#) la define como:

$$\text{Silueta}(i) = \frac{b_i - a_i}{\max(a_i, b_i)}$$

Donde b_i es la distancia mínima del conjunto de distancias medias de cada nodo i a los clusters que no pertenece, es decir, la distancia al clúster más cercano, a_i es la distancia media a los nodos del clúster al que pertenece i . Este coeficiente se acota en el intervalo $[-1,1]$ indicando, con valores positivos, que la asignación es correcta y con valores negativos que el nodo debe pertenecer al clúster más cercano.

Al igual que los coeficientes de cohesión y separación, podemos calcular el valor local del clúster de i o el valor global de la red calculando la media de esta medida:

$$\text{Silueta}(C) = \frac{1}{|C|} \sum_{i \in C} \text{Silueta}(i)$$

O para toda la red completa:

$$\text{Silueta}(\pi) = \frac{1}{|\pi|} \sum_{C \in \pi} \text{Silueta}(C)$$

Esta medida obtiene su mejor valor cuando la cohesión dentro de un clúster es mínima. Esta medida plantea problemas en redes donde existan grupos de nodos aislados ya que no se podrá calcular correctamente puesto que no podremos calcular la distancia entre clusters.

3.5.5. Coeficiente de Cobertura

Uno de los requerimientos de la detección de comunidades es que dichas comunidades estén densamente conectadas internamente y débilmente conectadas entre sí. El **coeficiente de cobertura**, usado por [Brandes et al. \(2003\)](#), cuantifica el peso de aristas dentro de un clúster frente al peso global del grafo. Formalmente se define como:

$$\text{Cobertura}(C) = \frac{W(C)}{W(G)}$$

Para calcular el valor global basta con sumar los valores locales, es decir:

$$\text{Cobertura}(\pi) = \sum_{C \in \pi} \text{Cobertura}(C)$$

Este valor global nos viene a decir el porcentaje de aristas que están dentro de clusters, es decir, mientras menos aristas tengamos uniendo clusters mejor resultado obtendremos. Uno de los problemas de esta medida es que tiende a favorecer asignaciones con pocos clusters, ya que el número de aristas que los conectan será menor.

3.6. Diseño de la implementación

Para terminar con esta sección, mostraremos los principales diagramas de clases que plasman el diseño jerárquico que hemos planteado para desarrollar las técnicas y medidas descritas anteriormente.

3.6.1. Algoritmos

Existe una superclase para englobar a todos los algoritmos, llamada **CDAgorithm**. Cada familia tiene asociada una clase padre, cuyas hijas implementan las diferentes variantes de la familia.

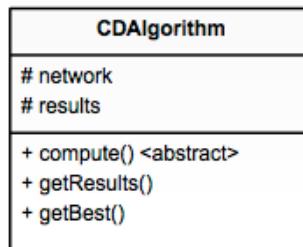


Diagrama 1. Superclase CDAgorithm

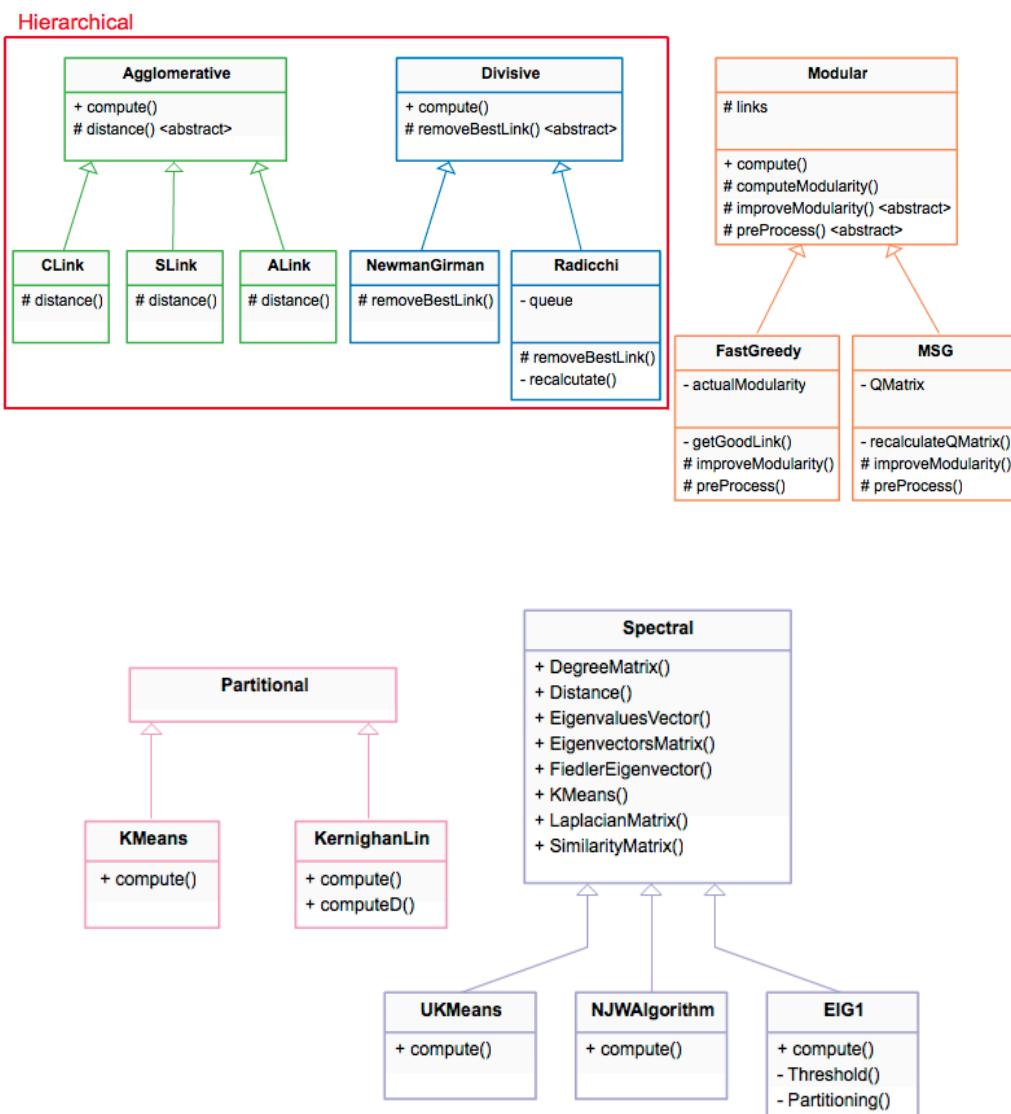


Diagrama 2. Clases Algoritmos

3.6.2. Medidas

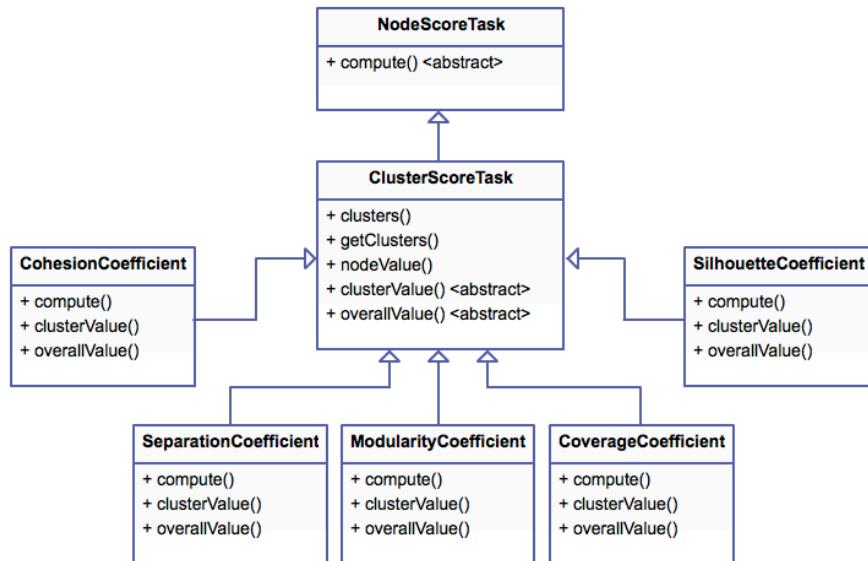


Diagrama 3. Clases Medidas

4. Implementación

Los algoritmos han sido implementados en *Java*, como ampliación de un proyecto del que forma parte nuestro tutor, Fernando Berzal Galiano. Se trata del proyecto *NOESIS*, una herramienta para el análisis y visualización de redes. El código fuente se puede consultar sin restricción alguna ya que está liberado en un repositorio de Google (<https://code.google.com/p/cdws/>).

4.1. Paralelismo

Para aprovechar todo el potencial de los procesadores con varios núcleos de ejecución, sistemas distribuidos y la posibilidad de paralelismo en algunos cálculos, se ha usado el framework *Parallel* de *NOESIS*. Esto nos permite lanzar de forma cómoda e independiente, diferentes cálculos que forman parte de un mismo resultado, para ser fusionados al final.

El modelo que se sigue en la parallelización es el de *MapReduce*, usado inicialmente por Google para optimizar su algoritmo [PageRank](#). [Dean y Ghemawat \(2008\)](#), ingenieros de Google, muestran una serie de experimentos muy interesantes donde muestran, por ejemplo, la ordenación de 10^{10} elementos de 100 bytes, lo que equivale a 1 Terabyte de información. *MapReduce* se basa en 2 operaciones básicas:

- *Map*: encargada de la partición del problema en subproblemas.
- *Reduce*: fusión de los resultados parciales.

En la siguiente ilustración mostramos un esquema de su funcionamiento, además encontramos un paso intermedio (*shuffle*) correspondiente al reparto de tareas.

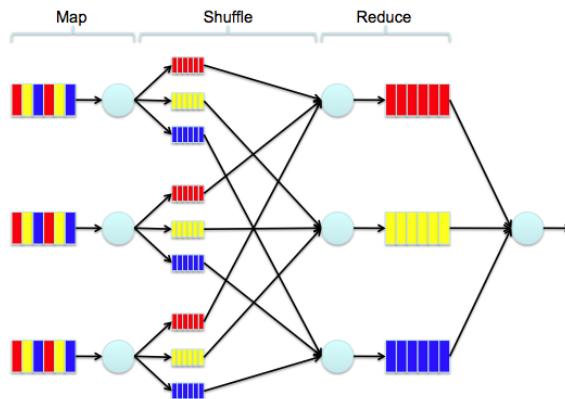


Diagrama 4. Paralelismo MapReduce (Fuente: SlideShare)

5. Pruebas

Con el objeto de evaluar tanto la calidad como la eficiencia de los métodos estudiados en este proyecto, se ha realizado y analizado una completa batería de pruebas. En esta sección vamos a plasmar los resultados acompañados de algunas gráficas que nos faciliten su entendimiento y justifiquen la teoría que explicamos en la descripción de los algoritmos.

5.1. Pruebas de Calidad

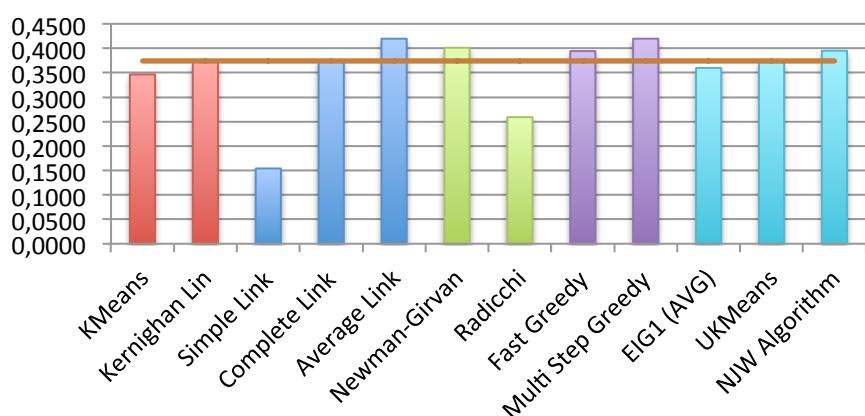
Hemos tomado varias redes típicas para test, además de dos casos reales en la red de *Facebook*. Las características de cada una y sus resultados obtenidos en las medidas de evaluación se muestran a continuación.

También se añade una gráfica de la modularidad ya que es la medida más informativa acerca de la calidad de los resultados, incluida una línea orientativa que marca el valor medio para tenerlo de referencia. (Nótese que para esta media se ha obviado el resultado del Algoritmo Aglomerativo Simple Link debido a sus pésimo funcionamiento).

Club de Karate: 34 Nodos, 78 enlaces.

Algoritmo	T (ms)	Clusters	Modularidad	Cohesión	Separación	Silueta	Cobertura
KMeans	3	4	0,3466	17,0588	62,4118	0,1339	0,6538
Kernighan Lin	8	2	0,3718	30,1765	49,2941	0,3289	0,8718
Simple Link	14	17	0,1541	17,1176	62,3529	0,4036	0,4615
Complete Link	7	3	0,3747	22,4706	57,0000	0,2894	0,7949
Average Link	8	4	0,4198	13,8235	65,6471	0,2317	0,7308
Newman-	116	5	0,4013	12,8824	66,5882	0,1956	0,6923
Radicchi	22	12	0,2593	13,3529	66,1176	0,3397	0,6410
Fast Greedy	49	3	0,3944	21,7647	57,7059	0,2503	0,8077
Multi Step	77	4	0,4198	13,8235	65,6471	0,2317	0,7308
EIG1 (AVG)	3	2	0,3600	30,2353	49,2353	0,3458	0,8718
UKMeans	1	2	0,3715	29,8235	49,6471	0,3473	0,8718
NJW Algorithm	3	4	0,3951	13,7647	65,7059	0,1848	0,6667

Tabla 1. Pruebas Calidad (Karate)

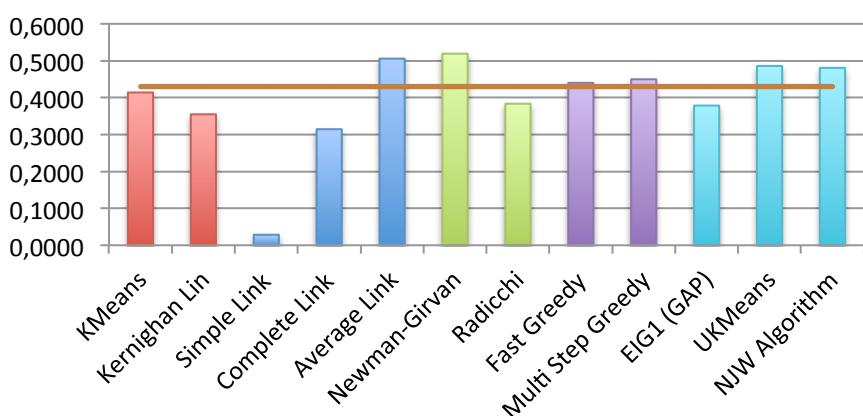


Gráfica 1. Modularidad (Karate)

Familias de delfines: 62 Nodos, 159 enlaces.

Algoritmo	T (ms)	Clusters	Modularidad	Cohesión	Separación	Silueta	Cobertura
KMeans	7	4	0,4141	40,7742	164,0000	0,1989	0,7421
Kernighan Lin	19	2	0,3550	90,5484	114,2258	0,1686	0,8553
Simple Link	34	34	0,0285	41,3871	163,3871	0,2404	0,3019
Complete Link	20	21	0,3146	4,7097	200,0645	0,2440	0,4151
Average Link	19	7	0,5059	21,2258	183,5484	0,2249	0,7170
Newman-	397	5	0,5194	32,5484	172,2258	0,2876	0,7987
Radicchi	33	26	0,3837	8,7419	196,0323	0,3026	0,5597
Fast Greedy	140	5	0,4404	41,1290	163,6452	0,2128	0,7736
Multi Step	169	4	0,4500	35,2258	169,5484	0,2393	0,7170
EIG1 (GAP)	6	2	0,3787	80,8710	123,9032	0,4464	0,9623
UKMeans	3	5	0,4860	23,3871	181,3871	0,2097	0,6981
NJW Algorithm	13	5	0,4809	24,1290	180,6452	0,1863	0,7044

Tabla 2. Pruebas Calidad (Delfines)

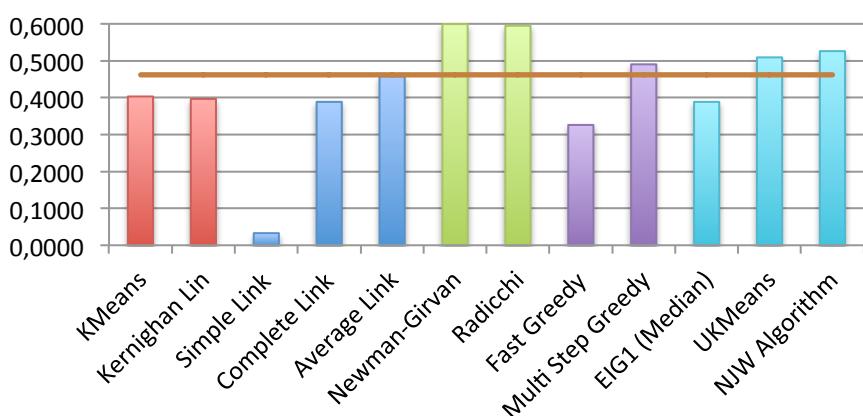


Gráfica 2. Modularidad (Delfines)

Equipos de Fútbol y cruces entre ellos: 115 Nodos, 613 enlaces.

Algoritmo	T (ms)	Clusters	Modularidad	Cohesión	Separación	Silueta	Cobertura
KMeans	67	4	0,4033	71,6522	214,2783	0,0830	0,6949
Kernighan Lin	97	2	0,3967	125,5652	160,3652	0,1970	0,8972
Simple Link	77	70	0,0328	44,4000	241,5304	0,3626	0,2023
Complete Link	63	7	0,3887	40,4348	245,4957	0,1001	0,5791
Average Link	53	5	0,4577	47,5652	238,3652	0,1726	0,6803
Newman-	2067	10	0,5996	16,1391	269,7913	0,3011	0,7096
Radicchi	109	13	0,5952	10,6609	275,2696	0,3402	0,6835
Fast Greedy	419	8	0,3262	45,9478	239,9826	-0,0016	0,5334
Multi Step	469	4	0,4904	65,0957	220,8348	0,1941	0,7847
EIG1 (Median)	116	2	0,3887	125,3043	160,6261	0,2002	0,8891
UKMeans	14	5	0,5093	40,6783	245,2522	0,2189	0,7129
NJW Algorithm	27	4	0,5262	54,5043	231,4261	0,2040	0,7814

Tabla 3. Pruebas Calidad (Fútbol)

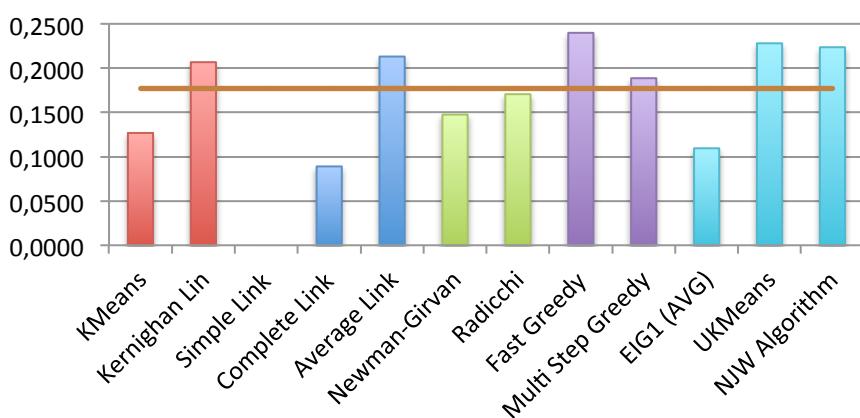


Gráfica 3. Modularidad (Fútbol)

Red de amigos en *Facebook* de Francisco: 137 Nodos, 1336 enlaces.

Algoritmo	T (ms)	Clusters	Modularidad	Cohesión	Separación	Silueta	Cobertura
KMeans	17	20	0,1269	14,3796	238,1168	-0,0123	0,2732
Kernighan Lin	140	2	0,2068	119,0949	133,4015	0,0936	0,8159
Simple Link	107	1	0,0000	252,4964	0,0000	-1,0000	1,0000
Complete Link	143	49	0,0890	4,0730	248,4234	0,2075	0,1639
Average Link	159	21	0,2132	23,1533	229,3431	0,1616	0,4251
Newman-Girvan	6658	38	0,1476	58,6423	193,8540	0,1978	0,7732
Radicchi	601	30	0,1707	48,3358	204,1606	0,1536	0,7448
Fast Greedy	1547	5	0,2399	61,4015	191,0949	0,0649	0,5494
Multi Step Greedy	2648	4	0,1887	100,5109	151,9854	0,0812	0,7163
EIG1 (AVG)	65	2	0,1095	194,6861	57,8102	0,0821	0,9102
UKMeans	12	9	0,2281	38,8905	213,6058	0,0960	0,5015
NJW Algorithm	23	10	0,2237	22,2774	230,2190	0,1090	0,3787

Tabla 4. Pruebas Calidad (*Facebook* Francisco)

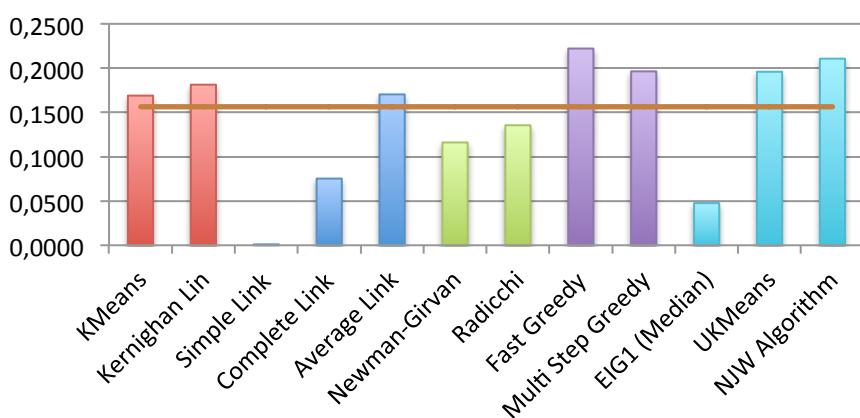


Gráfica 4. Modularidad (*Facebook* Francisco)

Red de amigos en *Facebook* de Aarón: 270 Nodos, 3277 enlaces.

Algoritmo	T (ms)	Clusters	Modularidad	Cohesión	Separación	Silueta	Cobertura
KMeans	56	23	0,1690	45,9481	467,7778	-0,0114	0,3781
Kernighan Lin	815	2	0,1814	246,8074	266,9185	0,0685	0,8731
Simple Link	393	128	0,0010	143,9630	369,7630	0,2392	0,2658
Complete Link	933	116	0,0753	2,7852	510,9407	0,2886	0,1147
Average Link	346	56	0,1704	25,4000	488,3259	0,1891	0,3506
Newman-	33207	105	0,1161	71,7333	441,9926	0,2630	0,7208
Radicchi	1467	72	0,1355	88,8519	424,8741	0,1314	0,7205
Fast Greedy	11021	9	0,2222	144,2815	369,4444	0,0318	0,4565
Multi Step	84166	6	0,1964	151,1185	362,6074	0,0396	0,5581
EIG1 (Median)	224	2	0,0476	254,0222	259,7037	0,0147	0,5758
UKMeans	117	7	0,1960	97,5111	416,2148	0,0548	0,4791
NJW Algorithm	235	12	0,2108	46,4444	467,2815	0,0773	0,3741

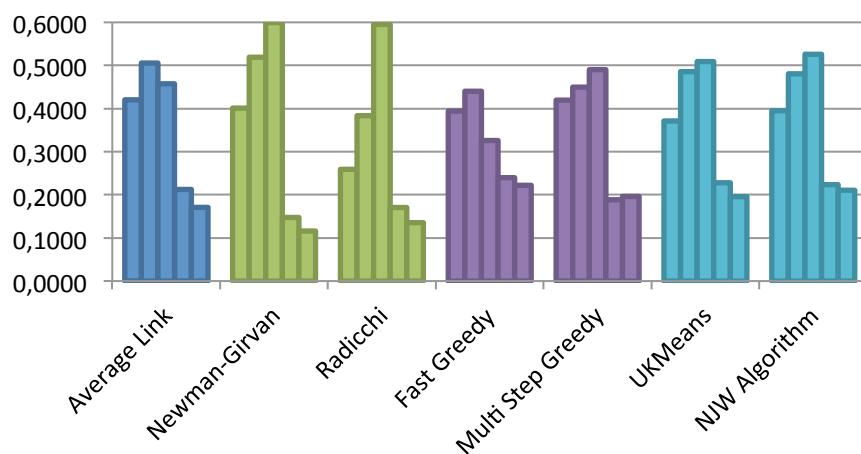
Tabla 5. Pruebas Calidad (*Facebook* Aarón)



Gráfica 5. Modularidad (*Facebook* Aarón)

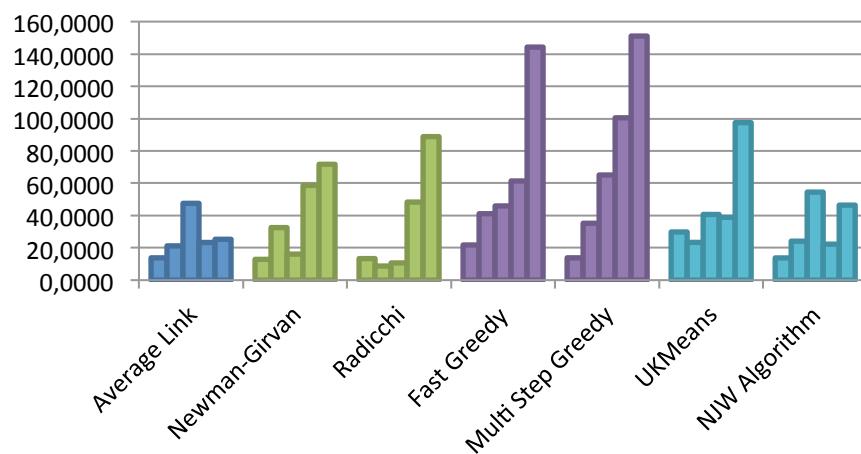
Para tener una vista global de cómo se comportan cada medida en las diferentes redes, se incluyen las siguientes gráficas. Nos hemos quedado con los mejores algoritmos para una mayor claridad, de modo que, para cada medida veremos en una gráfica como se ha comportado cada algoritmo en cada una de las redes. Las redes se muestran en el mismo orden en el que se han plasmado anteriormente (Karate, Delfines, Fútbol, *Facebook* Aarón, *Facebook* Francisco).

Modularidad.



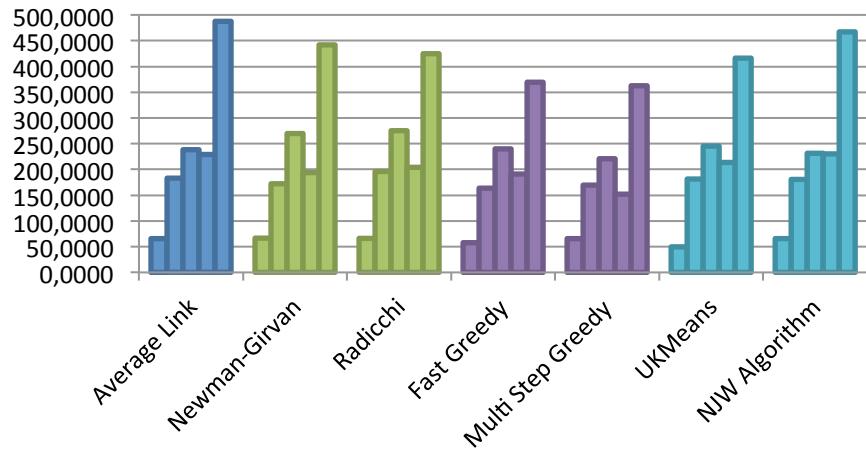
Gráfica 6. Modularidad (global)

Cohesión.



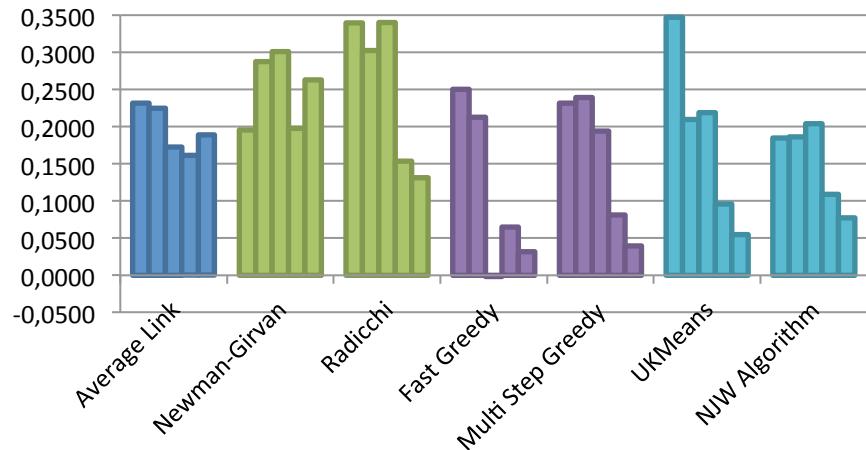
Gráfica 7. Cohesión (global)

Separación.



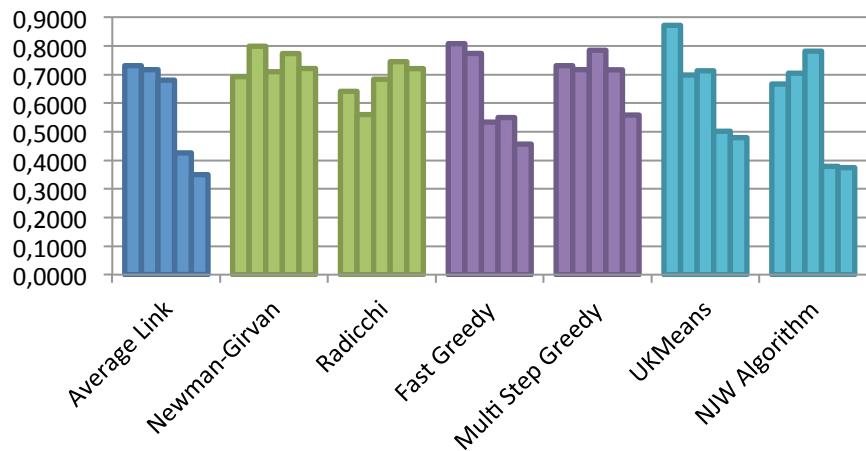
Gráfica 8. Separación (global)

Silueta.



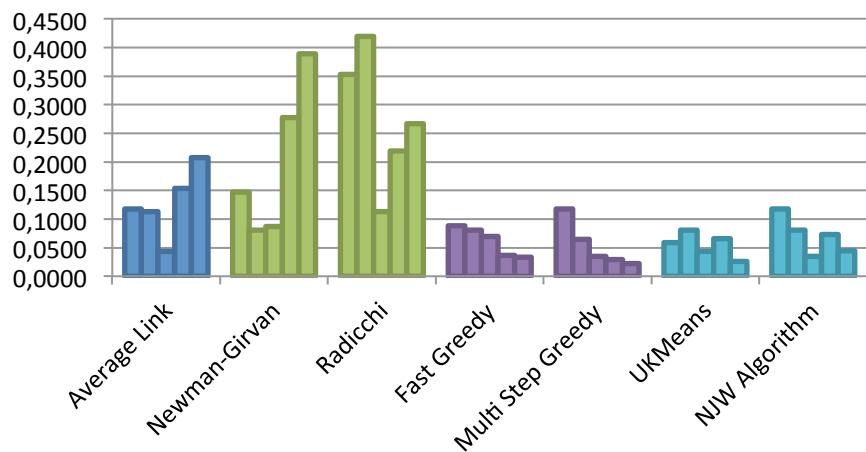
Gráfica 9. Silueta (global)

Cobertura.



Gráfica 10. Cobertura (global)

Número de Clusters Normalizado.



Gráfica 11. Número de Clusters Normalizado (global)

5.2. Pruebas de Tiempo

Primero, la tabla de tiempos obtenidos en las pruebas de eficiencia. Para construir este conjunto de pruebas, hemos ido generando redes de cliques unidos entre sí, con diferentes números de nodos y enlaces, para después aplicarle todos y cada uno de los algoritmos implementados y ver como se comportan, en tiempo, conforme el tamaño del problema crece.

La primera fila cabecera, nos informa del número de nodos, seguido debajo, del número de enlaces de la red en cuestión. Por ejemplo $\frac{100}{460}$ se refiere a una red de 100 nodos y 460 enlaces.

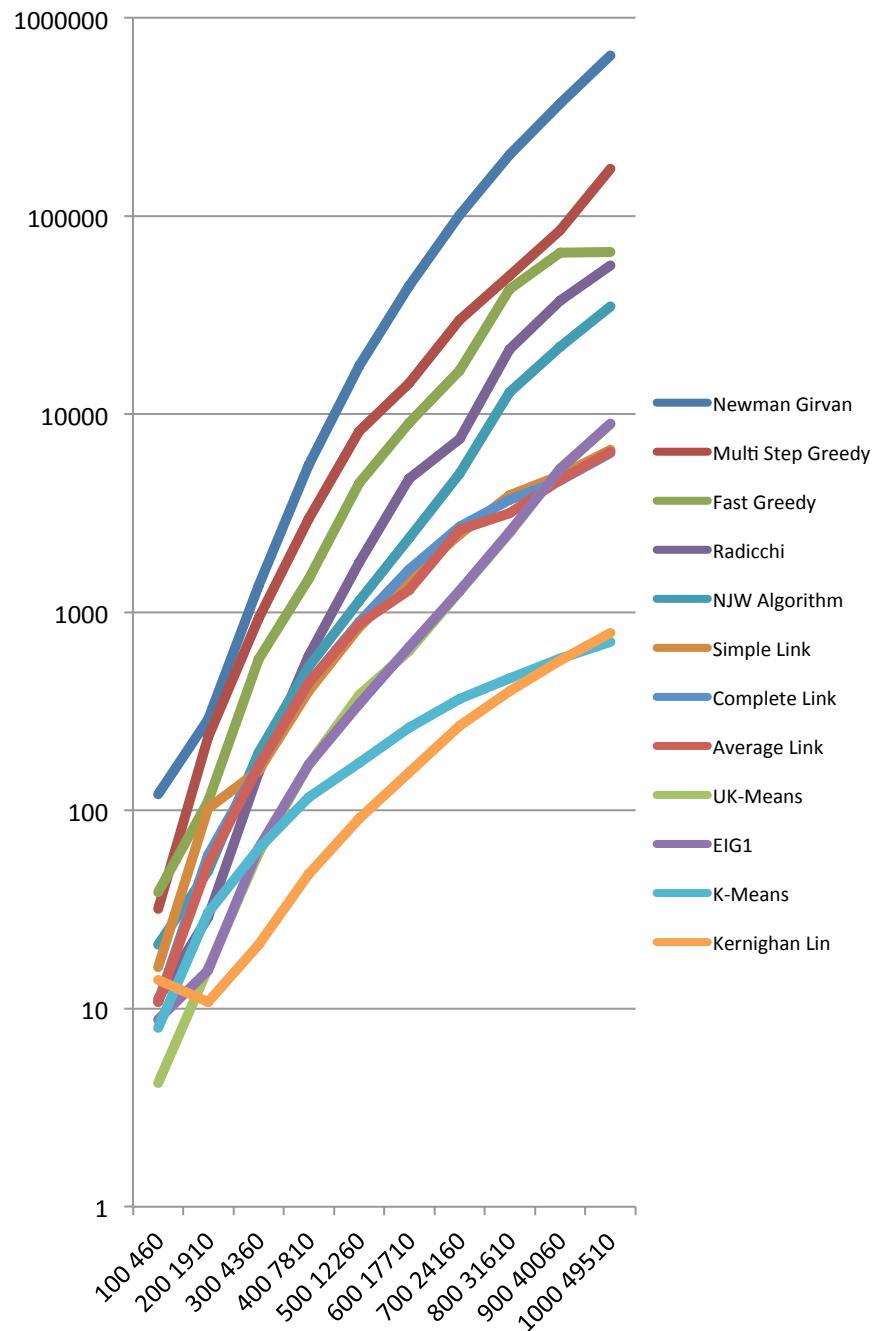
Los tiempos están medidos en **milisegundos**.

Algoritmo	100 460	200 1910	300 4360	400 7810	500 12260	600 17710	700 24160	800 31610	900 40060	1000 49510
Newman Girvan	120,6	289,4	1361,4	5523,2	17526,4	44536,8	101656	203693	371282	646056,6
Multi Step Greedy	31,8	235	934,2	2953,2	8205,4	14318,2	29830,2	49895,2	84559,6	173531,2
Fast Greedy	38,6	111,4	581,8	1468,6	4500,4	9044	16638	42891,4	65403,8	65758,6
Radicchi	11,2	29	156	603,8	1779,6	4720,8	7499,2	21223,2	37451,6	56334,6
NJW	21	49,4	195,2	530,8	1145,6	2371,4	5029,4	12866,4	21834,4	35078,6
Simple Link	16,2	102,2	159,4	400,8	842	1487	2473,8	3887,4	4893	6591,2
Complete Link	10,8	59,4	169,4	439,4	891	1648,8	2703,8	3707,8	4639	6384
Average Link	10,8	53,6	167,4	441,2	873,8	1291,8	2605,8	3168	4673,8	6452
UK-Means	4,2	16	61	169,6	383	641,2	1280,8	2510,8	5322,4	8894,8
EIG1	8,8	15,6	64,8	170,2	344,6	668	1286,2	2553	5299,6	8972,2
K-Means	8	30,4	64	116,6	174,4	261,4	366,2	463,2	585,2	707,8
Kernighan Lin	14	10,8	21	47,6	90,8	155,4	267,8	400,8	570	786,6

Tabla 6. Pruebas de Tiempo

En la siguiente página, para verlo con más claridad, volcaremos los resultados de esta batería de pruebas sobre un gráfica en escala logarítmica (debido a los altos valores que presentan las técnicas computacionalmente más costosas).

En la leyenda, se relaciona cada color con el algoritmo correspondiente. Se pueden ver ordenadas de mayor a menor, según el tiempo de ejecución.



Gráfica 12. Pruebas de Tiempo

6. Aplicaciones en Redes Sociales

La Detección de Comunidades en Redes Complejas es una tarea directamente aplicable a las Redes Sociales que manejamos habitualmente. La idea más básica puede ser agrupar a los usuarios que se conectan con nosotros en función de sus relaciones entre ellos. Un ejemplo de esta utilidad puede apreciarse en la siguiente ilustración, fruto de realizar el clustering en una red de amigos de *Facebook* con el Algoritmo de *Newman-Girvan* y posteriormente visualizar la red con la herramienta *NOESIS*.

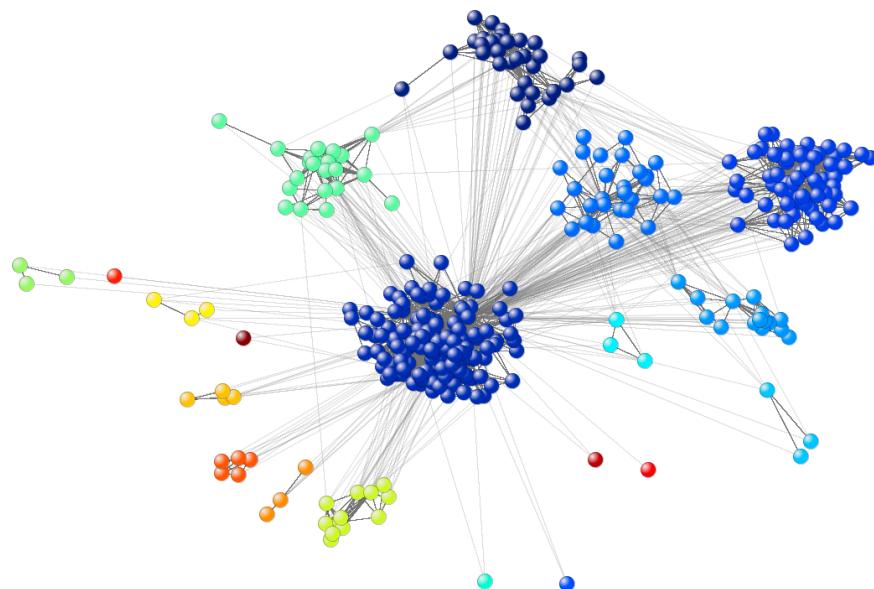


Ilustración 3. Detección de Comunidades en una red de Facebook

Explorando el grafo anterior y viendo algunos nombres de los integrantes de cada grupo es fácil identificar cada clúster: familia, universidad, instituto, amigos del pueblo, amigos del pádel, etc.

Automatizar este proceso puede complicarse si queremos emplear nuestros datos en las redes sociales más conocidas. Sus políticas de privacidad y uso de la información nos restringen bastante; situación en la que nos vimos envueltos cuando estudiamos esta posibilidad. A modo de resumen, mostraremos la siguiente tabla que material ofrece cada red social.

Red Social	Enlaces Nivel 1	Enlaces Nivel 2
<i>Facebook</i>	Sólo de los que usen la app.	Sólo si ambos usuarios usan la app.
<i>LinkedIn</i>	Todos	Ninguno
<i>Twitter</i>	Todos (límite datos/tiempo)	Todos (límite datos/tiempo)

Tabla 7. Permisos Redes Sociales

Dado que para conseguir un resultado como el anterior, necesitamos enlaces de primer (el usuario con sus amigos) y segundo nivel (sus amigos con sus amigos), no es posible realizar esta tarea de forma automática: *Facebook* solo nos da información de nuestros amigos y sus enlaces sólo si dichos usuarios utilizaran y dieran permiso a nuestra aplicación de *Facebook*, *LinkedIn* solo nos da los enlaces de primer nivel y *Twitter* si que nos da lo que necesitamos pero no podemos obtenerlo todo en un tiempo razonable porque nos limita la cantidad de datos a recibir en períodos de tiempo.

Esto nos lleva a pensar que por razones de interés privado, las redes sociales no están dispuestas del todo a devolvernos nuestros datos para que los usemos en otra aplicación. Por otra parte también es lógico ya que son los propietarios de esta información en cuánto a su uso fuera de la red social.

Sin embargo, esta aplicación de detección de comunidades, siendo cuánto menos curiosa, no es a la que más partido pueda sacársela, en principio, de cara al usuario estándar. Usando un poco la imaginación y analizando los medios que teníamos surgieron nuevas ideas más funcionales y novedosas que dieron lugar a nuestro aplicación *Wisper*.

7. Wisper



En esta sección vamos a presentar la cara visible de nuestro trabajo en la investigación e implementación de algoritmos orientados a la Detección de Comunidades en Redes. Dados los tiempos que corren es casi inmediato pensar en Redes Sociales cuando uno se plantea posibles casos prácticos donde aplicar técnicas de clustering para redes complejas; así, Wisper (wisper.es) surge para dar una nueva funcionalidad en cuanto a exploración de los contenidos compartidos en Redes Sociales como *Twitter*, *Facebook*, *Google Plus* o *LinkedIn*.

7.1. Una herramienta común para diferentes Redes Sociales

Con el objetivo de ofrecer una herramienta general capaz de trabajar con cualquier red social estándar de las que manejamos habitualmente, Wisper se ha diseñado para ser capaz de aplicar los algoritmos de detección en diferentes redes, gracias a el proceso de abstracción que hemos llevado a cabo para tratarlas todas de forma general: usuarios, conexiones y contenidos.

7.2. ¿Para qué sirve?

Wisper ofrece una nueva forma de ver qué se comparte en nuestros perfiles sociales de Internet. Su función es explorar los usuarios con los que estamos conectados, agruparlos y darnos la posibilidad de consultar los contenidos de forma segmentada según los conjuntos detectados. En definitiva, una herramienta que, en tiempo real, nos proporciona una manera más ordenada de ver los asuntos, imágenes y enlaces, entre otros, que se comentan en nuestro perfil.

Un ejemplo, si un grupo de amigos esta hablando sobre un tema, Wisper formará un grupo en base a sus conexiones y podremos ver todos los mensajes intercambiados entre ellos de forma aislada e inmediata.

La aplicación también tratará a los usuarios con los que nuestros amigos interactúen de modo que podremos identificar candidatos a posibles amigos si no estamos conectados con ellos explícitamente.

Si queremos ir más allá, incluso tenemos la opción de escoger un grupo e identificar subgrupos dentro de él, lo cual puede ser interesante cuando se presenten conjuntos con muchos usuarios .

Gracias a esto, no tendremos que ver todos los asuntos seguidos sin orden aparente, podremos dividirlo según los grupos de usuarios que comparten los contenidos, tal y como hacen las listas en *Twitter* o en *Facebook*, con la ventaja de que estas listas se determinan automática y dinámicamente según la actividad reciente.

Además, esta aplicación es capaz de realizar las acciones más típicas y destacadas dentro de la red social, ofreciendo al usuario la posibilidad de interactuar desde el propio sistema.

7.3. ¿Cómo funciona?

Wisper es una aplicación web alojada en wisper.es con un diseño adaptativo para ser correctamente visualizada en las diferentes pantallas que nos podemos encontrar hoy día, desde ordenadores, tabletas, hasta móviles. Para entender su funcionamiento, vamos a apoyarnos en un diagrama de secuencia.

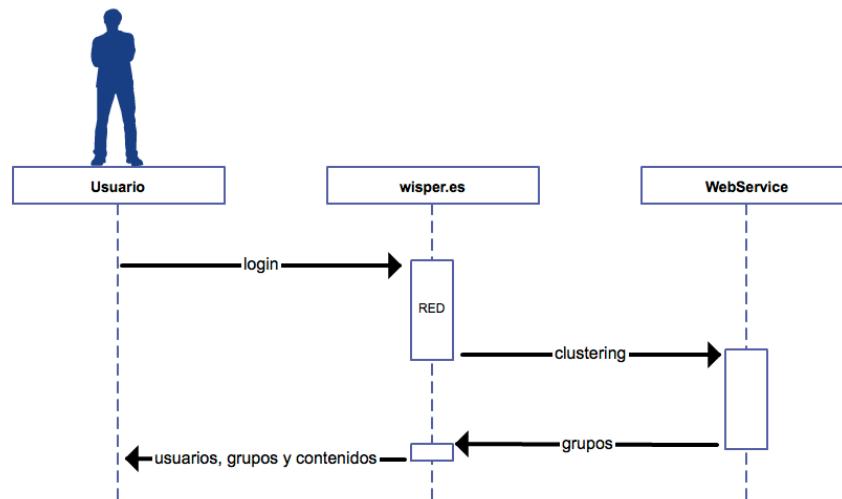


Diagrama 5. Secuencia Wisper

El usuario se loguea autorizando el uso de sus datos en la red social, el sistema los recoge y genera una red para ser tratada con los algoritmos implementados de Detección de Comunidades, posteriormente hace una petición al *Web Service* donde hemos alojada la aplicación que realiza el clustering y cuando recibe la respuesta construye la interfaz dinámica, propia de una aplicación de escritorio, para facilitársela al usuario y que este interactúe con ella.

La comunicación entre Wisper y el *Web Service* está optimizada para ser lo más ligera posible, se hace de la siguiente forma:

- Wisper envía en formato *CSV* el número de nodos que tiene la red y los enlaces que existen en ella.
- El *Web Service* responde también en formato *CSV*, la asignación de los nodos en cada clúster.

Como se ha mencionado, la interfaz esta diseñada con hojas de estilo *CSS3* aplicando la filosofía *RWD* (del inglés, *Responsive Web Design*) la cual permite una mejor experiencia del usuario cuando acceda desde diversos dispositivos. Las secciones y contenidos de la aplicación son redimensionados y situados convenientemente según el tamaño del display que lo muestre, dotando así de una mayor usabilidad y claridad en la interfaz.

Todo su control esta gestionado con *JavaScript* ejecutado en el cliente, apoyado por una de sus librerías, *jQuery*, y la técnica de desarrollo *AJAX* (*Asynchronous JavaScript And XML*); haciendo así esta web dinámica cual aplicación de escritorio, sin desagradables recargas de página. Una vez la aplicación es cargada, su interacción es fluida y rápida.

7.4.Trabajando con *Twitter*

Twitter es una práctica directa para Wisper ya que tenemos usuarios, conexiones (*Followers* y *Following*) y contenidos, en este caso *Tweets*. Basta con obtener los datos y la red para que empiece a funcionar. A continuación se comentará la forma de determinar una grafo según la actividad reciente de los usuarios que seguimos, además de explicar las acciones que ofrece Wisper para interactuar con este servicio de *microblogging* y hacer así, más potente nuestra aplicación en cuanto a funcionalidades.

7.4.1.La red de nuestro Timeline

Tras el análisis de los *tweets* y las formas que ofrecen para conectar usuarios, se ha determinado un procedimiento para extraer una red, correspondiente a las relaciones entre usuarios según sus contribuciones. Explicamos como:

- Un usuario que *retwittea* un *tweet*, cabe considerarlo como relacionado con el autor original de dicho *tweet*.

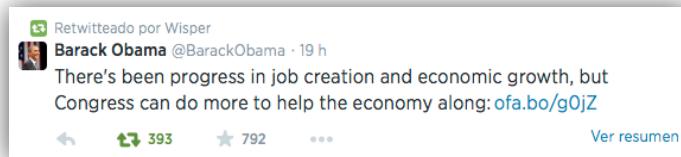


Ilustración 4. Retweet

- Si un usuario responde al *tweet* de otro, podemos valorar que entre ellos dos hay una conexión debido a la conversación que se forma.



Ilustración 5. Respuesta Tweet

- Cuando tenemos una serie de entidades mencionadas en un mismo *tweet* es lógico pensar que entre ellas existen vínculos que los enlazan a todos.



Ilustración 6. Mención Tweet

- Los usuarios que estén utilizando un mismo *hashtag* para etiquetar sus *tweets* es porque están hablando del mismo tema y por tanto interesa tener en cuenta este nexo.



Ilustración 7. Hashtag

En base a esto, recuperando los *tweets* más recientes que se comparten en nuestro perfil obtenemos un conjunto usuarios y una serie de relaciones inferidas que nos generan una red susceptible de ser sometida a algún algoritmo de detección de comunidades. Si a además, tenemos los *tweets* que han dado lugar a la red, podremos utilizar el conocimiento extraído con los algoritmos de detección para filtrarlos y consultar lo que nos interese en cada momento, que no es otro si no el cometido de Wisper.

7.5. Diseño e implementación

Sin ánimo de hacer de esta sección una lectura tediosa, nos centraremos en exponer a grandes rasgos la arquitectura general del sistema, cómo se descompone en módulos y la responsabilidad de cada uno.

El diseño se rige por el patrón de arquitectura software *MVC* (modelo-vista-controlador) el cual está claramente dividido en dos lugares donde se lleva a cabo:

- la vista (interfaz de usuario) se trata en el cliente gracias al intérprete de *JavaScript* instalado en los navegadores actuales y el diseño visual *HTML-CSS*.
- el modelo y controlador, es decir, lo referido a la obtención y manipulación de datos, es llevado a cabo en un servidor web con intérprete *PHP* (el mismo que sirve al cliente la vista).

Por otro lado, tenemos el *Web Service* donde se hospedan todos los algoritmos de Detección de Comunidades que han sido desarrollados para este proyecto. Dado que partimos del proyecto *NOESIS* de nuestro tutor, Fernando Berzal Galiano, su implementación ha sido escrita en *Java*. Este servicio nos proporcionará respuesta cuando solicitemos la detección de grupos dentro de una red.

7.5.1. Interfaz de usuario

Como se comenta anteriormente, la vista de la aplicación web esta gestionada mediante *JavaScript* ejecutado en el navegador cliente. Esto esta compuesto por dos módulos principales: elementos y controlador.

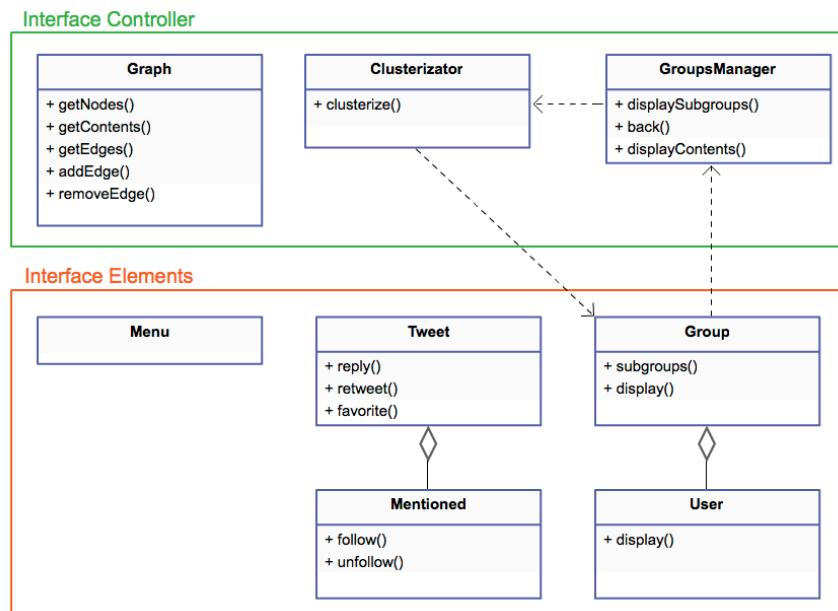


Diagrama 6. Clases Interfaz

El modulo **Controlador de Interfaz** se encarga de construir y gestionar la interfaz. Dentro él se hallan los siguientes tres componentes:

- **Graph:** encargado de la obtención de los datos a tratar. Conecta con el servidor que recoge los contenidos, usuarios y enlaces de la sesión. La recopilación de estos datos la hace el servidor de la aplicación web, cuya arquitectura se explica más adelante.
- **Clusterizator:** realiza la petición de Detección de Comunidades para agrupar a los usuarios que constituyen la red.
- **GroupsManager:** gestiona la navegación por los diferentes grupos y subgrupos, filtrando los contenidos según corresponda.

Dentro del módulo **Elementos de Interfaz** se encuentran las clases que definen a los principales objetos que vemos en la interfaz: *tweet*, usuario mencionado, grupo, usuario y menú; los cuales tienen asociados eventos y acciones que permiten la interacción dinámica con Wisper y la red social.

7.5.2. Servidor Web

Como todo recurso web, la aplicación Wisper es transmitida al cliente por un servidor web, siendo este el que facilita al cliente toda la implementación *JavaScript* y diseño *HTML* y *CSS*. Pero además de ello, como se menciona en la sección anterior, es el servidor el que recopila y obtiene los datos a tratar por Wisper, es decir, recupera los usuarios, enlaces y contenidos de la sesión en cuestión y se los envía a la aplicación cliente para que los maneje en la vista.

El proceso de obtención de la red y contenidos es ejecutado por el intérprete *PHP* del servidor, concretamente para *Twitter*, se ha usado la *API TwitterOAuth* que facilita las peticiones y conexión con la red social tanto como para obtener o introducir datos. De nuevo nos apoyaremos en un diagrama explicativo para hacernos una idea del diseño software que contempla esta tarea.

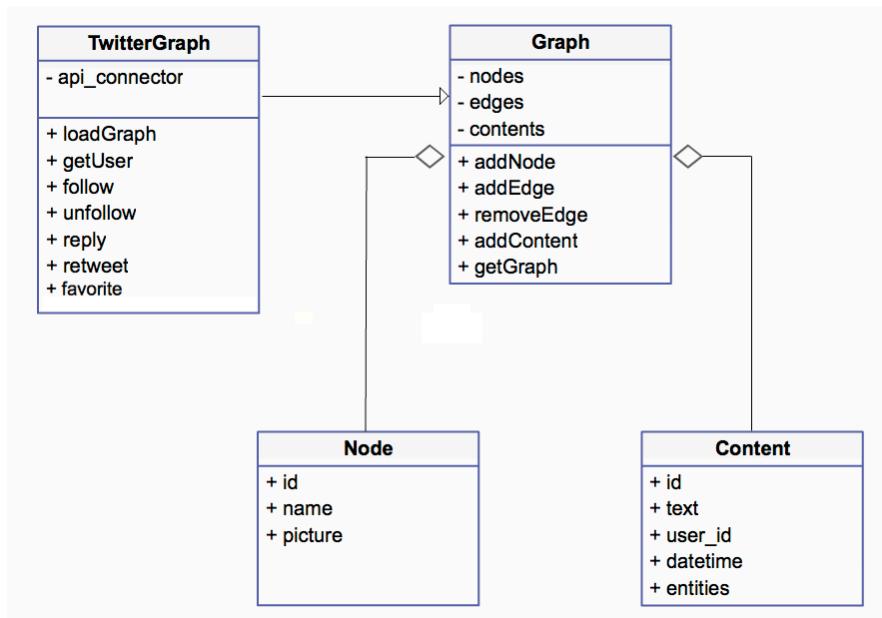


Diagrama 7. Clases Modelo/Controlador

Dentro de la arquitectura *MVC*, en primer lugar la parte del modelo englobaría a las clases **Nodo** y **Contenido**.

La clase **Grafo** o red, integra el conjunto de nodos, enlaces y contenidos. Es la superclase de la que heredarían sus hijas, propias de cada red social, como **TwitterGraph**.

Las clases hijas de Graph recogen los datos de la red social en cuestión y los vuelcan sobre el modelo, facilitándoselos a la vista, que los ofrecería al usuario en última instancia.

Dado este diseño, facilita la integración de Wisper con otras redes sociales. Bastaría con implementar la clase hija de Graph correspondiente, capaz de conectar con la red social e interactuar con ella.

7.5.3. Web Service

La aplicación que recibe las peticiones de realizar el clustering sobre las redes se encuentra alojada en la plataforma *OpenShift* propiedad de *RedHat*, que presta servicios de *cloud computing*. El *Webservice* implementa una interfaz tipo REST que responde al mismo conjunto de operaciones que el protocolo *HTTP*, el acceso al recurso se identifica mediante una única URI. En nuestro caso solamente se han implementado las operaciones *GET* y *POST* puesto que no almacenamos ninguna información, se recibe una red en formato *CSV* para minimizar el tráfico de red y se responde con la asignación de cada nodo a su clúster en el mismo formato. La plataforma *OpenShift* posee una gran carta de servidores de aplicaciones para *JAVA EE* como *GlassFish* o *JBOSS*. El problema que se nos planteó es que las versiones estables de las que disponía la plataforma de los servidores más comunes no son compatibles con la última versión de *Java* (v1.8), la cual usamos para la implementación del paquete de algoritmos por lo que investigamos servidores alternativos hasta que encontramos *WildFly*, para el cual disponíamos de su última versión (v8.1.0), en *OpenShift*, que es totalmente compatible con la versión 1.8 de *Java*. *WildFly* es un servidor de aplicaciones para *JAVA EE* basado en el conocido *JBOSS*.

7.5.3.1. Elección del Algoritmo.

Dado que disponemos de una remesa variada de algoritmos para la Detección de Comunidades, lo que hicimos para decantarnos por uno fue realizar una prueba de cómo se comportaban cada uno de ellos, con 4 usuarios distintos.

Los mejores métodos para este tipo de redes son:

- Jerárquico Aglomerativo Average-Link
- Jerárquico Divisivo Newman-Girvan
- Fast Greedy
- Multi Step Greedy

Otra aspecto a tener en cuenta es el tiempo de ejecución. Nos interesan técnicas buenas pero que también sean rápidas para una buena experiencia del usuario cuando use nuestra aplicación.

La técnica de Newman-Girvan es la mejor sin lugar a duda en este tipo de problema pero su tiempo de ejecución puede ser bastante elevado cuando en número de enlaces crece. Para ofrecer una respuesta más rápida, de las otras tres alternativas, optamos por el método Aglomerativo Average-Link ya que es el que nos da resultados más parecidos a Newman-Girvan en cuanto a modularidad y número de clusters, en un tiempo menor. Así que, en base a este análisis decidimos:

- Jerárquico Aglomerativo Average-Link para redes grandes.
- Jerárquico Divisivo Newman-Girvan para redes pequeñas.

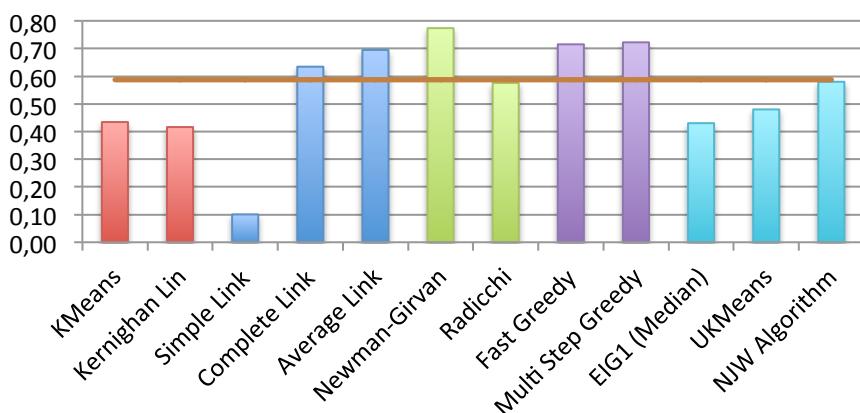
A continuación presentamos las tablas de resultados y gráficas asociadas, en el mismo formato que trabajamos en el capítulo Pruebas, para comprender y justificar este análisis.

Con el fin de proteger la privacidad de las personas que ofrecieron su perfil de *Twitter* para elaborar estas pruebas, los identificaremos anónimamente.

Usuario de Wisper 1

Algoritmo	T (ms)	Clusters	Modularidad	Cohesión	Separación	Silueta	Cobertura
KMeans	40	24	0,43	34,44	804,41	-0,02	0,50
Kernighan Lin	133	2	0,42	409,53	429,32	0,04	0,92
Simple Link	153	116	0,10	241,25	597,60	0,23	0,46
Complete Link	403	64	0,63	26,11	812,74	0,37	0,73
Average Link	406	49	0,70	23,60	815,25	0,35	0,78
Newman-Girvan	684	19	0,77	51,27	787,58	0,27	0,88
Radicchi	31	94	0,58	40,15	798,69	0,39	0,73
Fast Greedy	273	35	0,72	54,63	784,22	0,16	0,82
Multi Step Greedy	1126	39	0,72	32,32	806,53	0,24	0,81
EIG1 (AVG)	51	2	0,43	406,46	432,39	0,05	0,93
UKMeans	49	8	0,48	362,85	476,00	0,13	0,90
NJW Algorithm	121	8	0,58	254,09	584,76	0,15	0,89

Tabla 8. Pruebas Calidad (Usuario Wisper 1)

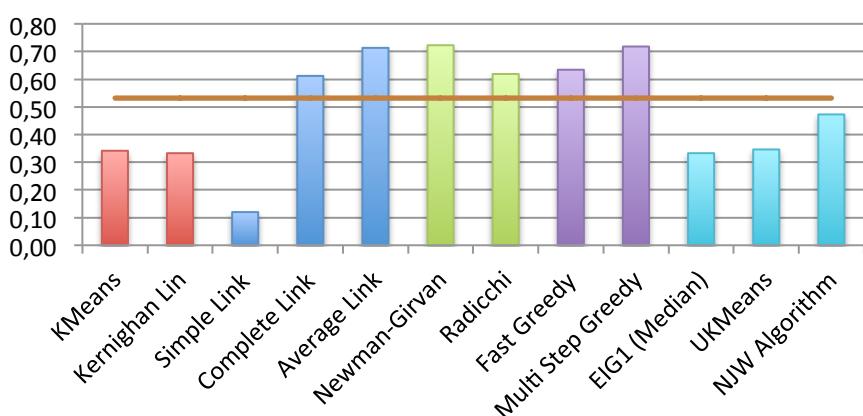


Gráfica 13. Modularidad (Usuario Wisper 1)

Usuario de Wisper 2

Algoritmo	T (ms)	Clusters	Modularidad	Cohesión	Separación	Silueta	Cobertura
KMeans	35	25	0,34	29,77	733,05	-0,04	0,42
Kernighan Lin	141	2	0,33	375,60	387,22	0,02	0,85
Simple Link	137	144	0,12	122,55	640,27	0,44	0,40
Complete Link	367	71	0,61	22,61	740,21	0,40	0,70
Average Link	346	35	0,71	55,85	706,97	0,36	0,85
Newman-Girvan	804	29	0,72	47,80	715,02	0,29	0,82
Radicchi	63	77	0,62	26,67	736,15	0,14	0,69
Fast Greedy	289	47	0,63	65,59	697,23	0,24	0,80
Multi Step Greedy	1762	32	0,72	65,91	696,91	0,23	0,85
EIG1 (AVG)	49	2	0,33	376,14	386,69	0,03	0,84
UKMeans	44	7	0,35	288,22	474,60	0,10	0,70
NJW Algorithm	96	5	0,47	291,07	471,75	0,01	0,81

Tabla 9. Pruebas Calidad (Usuario Wisper 2)

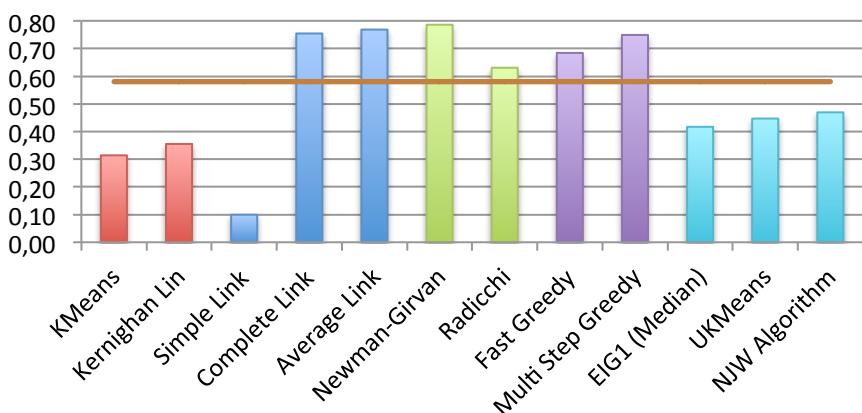


Gráfica 14. Modularidad (Usuario Wisper 2)

Usuario de Wisper 3

Algoritmo	T (ms)	Clusters	Modularidad	Cohesión	Separación	Silueta	Cobertura
KMeans	36	24	0,31	36,81	773,98	-0,08	0,38
Kernighan Lin	126	2	0,36	400,08	410,72	0,02	0,86
Simple Link	145	179	0,10	59,90	750,90	0,60	0,27
Complete Link	420	40	0,75	35,89	774,91	0,39	0,85
Average Link	401	27	0,77	72,65	738,15	0,36	0,89
Newman-Girvan	484	26	0,79	61,49	749,30	0,33	0,89
Radicchi	58	96	0,63	15,32	795,48	0,53	0,70
Fast Greedy	266	56	0,68	59,51	751,29	0,27	0,80
Multi Step Greedy	1510	43	0,75	39,51	771,29	0,29	0,83
EIG1 (AVG)	43	2	0,42	399,26	411,54	0,03	0,92
UKMeans	39	6	0,45	253,36	557,44	0,00	0,79
NJW Algorithm	102	6	0,47	229,14	581,66	0,03	0,76

Tabla 10. Pruebas Calidad (Usuario Wisper 3)

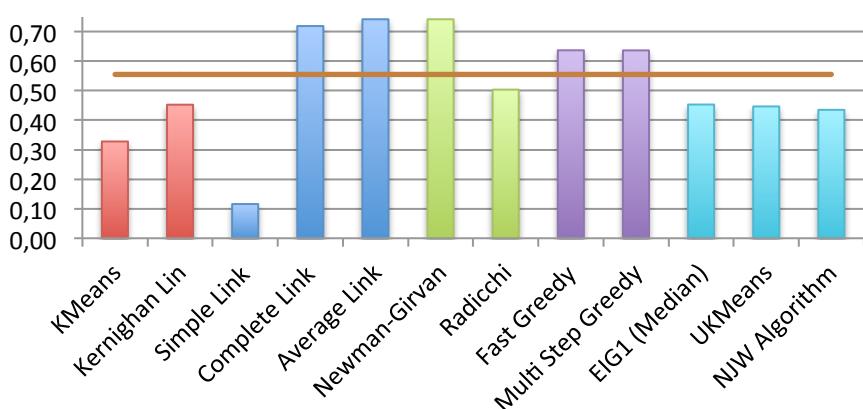


Gráfica 15. Modularidad (Usuario Wisper 3)

Usuario de Wisper 4

Algoritmo	T (ms)	Clusters	Modularidad	Cohesión	Separación	Silueta	Cobertura
KMeans	17	21	0,33	8,52	268,93	0,01	0,42
Kernighan Lin	19	2	0,45	124,45	153,00	0,16	0,95
Simple Link	25	56	0,12	28,52	248,93	0,51	0,34
Complete Link	26	12	0,72	22,19	255,26	0,34	0,90
Average Link	26	8	0,74	25,83	251,62	0,42	0,93
Newman-Girvan	92	8	0,74	25,83	251,62	0,42	0,93
Radicchi	11	36	0,50	8,76	268,69	0,53	0,64
Fast Greedy	36	20	0,64	15,95	261,50	0,28	0,78
Multi Step Greedy	74	21	0,64	12,90	264,55	0,31	0,77
EIG1 (AVG)	5	2	0,45	124,45	153,00	0,16	0,95
UKMeans	4	8	0,45	59,48	217,98	-0,03	0,74
NJW Algorithm	10	11	0,44	22,90	254,55	0,05	0,56

Tabla 11. Pruebas Calidad (Usuario Wisper 4)



Gráfica 16. Modularidad (Usuario Wisper 4)

7.6. Manual de Uso

Utilizar Wisper es muy fácil e intuitivo. Se compone por una barra de menú (donde se encuentran los botones para el manejo de la aplicación y los detalles del usuario que la utiliza) y dos zonas claramente diferenciadas: **Grupos** y **Contenidos**.



Ilustración 8. Wisper

El menú ofrece los botones comunes de Atrás, Actualizar y Cerrar Sesión, incluido un botón para mostrar los contenidos en caso de acceder a la aplicación desde un dispositivo móvil, donde no es posible mostrar ambas secciones a la vez (grupos y contenidos) debido al tamaño de la pantalla. Además, a la derecha encontramos los detalles más relevantes del usuario que usa la aplicación: su foto, nombre, número de posts y amigos, todo ellos vinculado con la página oficial correspondiente en la red social.

En la parte de los grupos, el usuario, puede navegar y ver que personas o entidades constituyen cada grupo. Si desea ver los contenidos de un grupo específico, basta con pinchar en él para que sean filtrados y visualizados solo aquellos en los que este interesado en ese momento.



Ilustración 9. Seleccionar un grupo en Wisper

De la misma forma podemos ver los contenidos referidos a un solo usuario en concreto, haciendo click sobre su foto en el grupo. La identificación de las fotos esta también facilitada por un *tooltip* que muestra el nombre del usuario cuando pasamos el ratón por encima.

Es posible que se den grupos de gran tamaño que el usuario quiera explorar más en profundidad, para ello disponemos del botón situado en la esquina inferior derecha del grupo. Así, Wisper aplicará de nuevo la Detección de Comunidades sobre ese grupo en concreto y nos mostrará los **subgrupos** resultantes. Esta acción podemos repetirla tantas veces como queramos mientras tengamos grupos un número de usuarios elevado. Para volver atrás después de haber explorado en detalle los subconjuntos dentro de un clúster, simplemente hay que pinchar en el botón correspondiente de la barra de menú.



Ilustración 10. Determinar subgrupos en Wisper

Para **interactuar con la red social**, iremos a la sección de contenidos. Aquí hallaremos todos lo relativo al grupo que este seleccionado con la posibilidad de realizar las acciones propias de la red social, tal y como estamos acostumbrados.

En caso de *Twitter*:

- *Retwittear un tweet.*
- Hacer o deshacer un *tweet* favorito.
- Responder a un *tweet*.
- Visitar el perfil de los usuarios presentes en el *tweet*, pinchando en su nombre.
- Seguir (*follow*) o dejar de seguir (*unfollow*) a un usuario, haciendo click sobre su foto parte inferior izquierda del *tweet*. La imagen se sombreará cuando no sigamos a esa persona y aparecerá clara cuando si lo hagamos.



Ilustración 11. Tweet en Wisper

8. Conclusiones y Trabajos Futuros

En este proyecto se ha abordado el problema de la Detección de Comunidades desde un punto de vista técnico. Hemos realizado una revisión de las principales familias de algoritmos que pueden emplearse para su resolución implementando varios métodos de cada una de ellas, lo que nos ha facilitado su comprensión. Además de poder evaluar estos métodos de forma teórica simplemente por sus eficiencias o por sus supuestas carencias o virtudes, hemos podido experimentarlas de forma práctica aplicándolos sobre un caso real de red compleja como es la red Social *Twitter*, lo que le ha dado mayor sentido a nuestro trabajo ya que, como futuros ingenieros, el poder abordar y resolver un problema real y actual es muy gratificante.

Otro punto a tener en cuenta es que la escalabilidad de estos métodos es reducida ya que la mayoría presentan un orden de eficiencia cúbico por lo que es necesario seguir mejorando las técnicas actuales y creando nuevas heurísticas que nos permitan resolver este problema, para grandes conjuntos de datos, en un tiempo reducido.

Cabe destacar la familia de los métodos espectrales ya que es una de las más recientes, además de que cada uno de sus componentes es un campo de estudio por si solo, como es la matriz laplaciana o los métodos de cálculo de valores y vectores propios. La robustez de sus técnicas han hecho que obtengan los mejores resultados en las redes reales, como las extraídas de la red personal de *Facebook*.

Como trabajo futuro sería muy interesante realizar un estudio de las técnicas más actuales para la detección de comunidades con solapamiento. Palla, Gergely, et al, 2005 indicaban que la mayoría de las redes reales se componen de comunidades anidadas y superpuestas, poniendo el ejemplo su vida personal (hobbies, familiares, etc.) y su vida científica. Es común que este tipo de algoritmos se basen en la detección de *k*-Cliques anidados, permitiendo esto que puedan compartir nodos. Los más conocidos son los algoritmos **CPM** cuyas iniciales se corresponden con **Métodos de Percolación de Cliques**.

De cara a la mejora de la aplicación Wisper y gracias a su diseño modular, nuestra idea es ampliar el número de redes sociales a consultar, introduciendo redes sociales como *Facebook*, *LinkedIn* o *Vine*, con la idea de poder gestionar toda la *vida social en internet* desde nuestra aplicación. El hecho de tener visitas desde EEUU e Inglaterra nos plantea la posibilidad de traducir nuestra aplicación a diferentes idiomas, objetivo que teníamos como secundario, aunque la aplicación está preparada para ello, y que ahora pasa a ser de primer nivel.

9. Bibliografía y listados

Bibliografía

- Biela, P., & Hamad, D. *Introduction to spectral clustering*.
- Brandes, U., Gaertler, M., & Wagner, D. (2003). *Experiments on graph clustering algorithms* (pp. 568-579). Springer Berlin Heidelberg.
- Chen, M., Nguyen, T., & Szymanski, B. K. (2013, September). *On measuring the quality of a network community structure*. In *Social Computing (SocialCom), 2013 International Conference on* (pp. 122-127). IEEE.
- Costa, L. D. F., Rodrigues, F. A., Travieso, G., & Villas Boas, P. R. (2007). *Characterization of complex networks: A survey of measurements*. *Advances in Physics*, 56(1), 167-242.
- Dean, J., & Ghemawat, S. (2008). *MapReduce: simplified data processing on large clusters*. *Communications of the ACM*, 51(1), 107-113.
- Defays, D. (1977). *An efficient algorithm for a complete link method*. *The Computer Journal*, 20(4), 364-366.
- Fortunato, S. (2010). *Community detection in graphs*. *Physics Reports*, 486(3), 75-174.
- Fortunato, S., & Barthelemy, M. (2007). *Resolution limit in community detection*. *Proceedings of the National Academy of Sciences*, 104(1), 36-41.
- Freeman, L. C. (1977). *A set of measures of centrality based on betweenness*. *Sociometry*, 35-41.
- Fruchterman, T. M., & Reingold, E. M. (1991). *Graph drawing by force-directed placement*. *Software: Practice and experience*, 21(11), 1129-1164.
- Hagen, L., & Kahng, A. B. (1992). *New spectral methods for ratio cut partitioning and clustering*. *Computer-aided design of integrated circuits and systems, ieee transactions on*, 11(9), 1074-1085.
- Kernighan, B. W., & Lin, S. (1970). *An efficient heuristic procedure for partitioning graphs*. *Bell system technical journal*, 49(2), 291-307.
- MacQueen, J. (1967, June). *Some methods for classification and analysis of multivariate observations*. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (Vol. 1, No. 14, pp. 281-297).
- Nascimento, M. C., & De Carvalho, A. C. (2011). *Spectral methods for graph clustering—A survey*. *European Journal of Operational Research*, 211(2), 221-231.

- Newman, M. E., & Girvan, M. (2004). *Finding and evaluating community structure in networks*. *Physical review E*, 69(2), 026113.
- Newman, M. E. (2005). *A measure of betweenness centrality based on random walks*. *Social networks*, 27(1), 39-54.
- Newman, M. (2010). *Networks: an introduction*. Oxford University Press.
- Ng, A. Y., Jordan, M. I., & Weiss, Y. (2002). *On spectral clustering: Analysis and an algorithm*. *Advances in neural information processing systems*, 2, 849-856.
- Palacio Niño, J. O. (2013). *Community Detection in Social Network*.
- Palla, G., Derényi, I., Farkas, I., & Vicsek, T. (2005). *Uncovering the overlapping community structure of complex networks in nature and society*. *Nature*, 435(7043), 814-818.
- Pattillo, J., Youssef, N., & Butenko, S. (2012). *Clique relaxation models in social network analysis*. In *Handbook of Optimization in Complex Networks* (pp. 143-162). Springer New York.
- Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., & Parisi, D. (2004). *Defining and identifying communities in networks*. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9), 2658-2663.
- Rousseeuw, P. J. (1987). *Silhouettes: a graphical aid to the interpretation and validation of cluster analysis*. *Journal of computational and applied mathematics*, 20, 53-65.
- Schuetz, P., & Cafisch, A. (2008). *Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement*. *Physical Review E*, 77(4), 046112.
- Sibson, R. (1973). *SLINK: an optimally efficient algorithm for the single-link cluster method*. *The Computer Journal*, 16(1), 30-34.
- Sokal, R. R. (1958). *A statistical method for evaluating systematic relationships*. *Univ Kans Sci Bull*, 38, 1409-1438.
- Von Luxburg, U. (2007). *A tutorial on spectral clustering*. *Statistics and computing*, 17(4), 395-416.

Listado de Ilustraciones

Ilustración 1. Transporte aéreo (Fuente: Northwestern University)	15
Ilustración 2. QMatrix (PHYSICAL REVIEW E 77, 046112 2008).....	44
Ilustración 3. Detección de Comunidades en una red de Facebook	70
Ilustración 4. Retweet	76

Ilustración 5. Respuesta <i>Tweet</i>	76
Ilustración 6. Mención <i>Tweet</i>	76
Ilustración 7. <i>Hashtag</i>	77
Ilustración 8. Wisper	86
Ilustración 9. Seleccionar un grupo en Wisper	87
Ilustración 10. Determinar subgrupos en Wisper.....	88
Ilustración 11. Tweet en Wisper	89

Listado de Gráficas

Gráfica 1. Modularidad (Karate).....	59
Gráfica 2. Modularidad (Delfines).....	60
Gráfica 3. Modularidad (Fútbol)	61
Gráfica 4. Modularidad (<i>Facebook</i> Francisco)	62
Gráfica 5. Modularidad (<i>Facebook</i> Aarón).....	63
Gráfica 6. Modularidad (global)	64
Gráfica 7. Cohesión (global)	64
Gráfica 8. Separación (global)	65
Gráfica 9. Silueta (global)	65
Gráfica 10. Cobertura (global)	66
Gráfica 11. Número de Clusters Normalizado (global)	66
Gráfica 12. Pruebas de Tiempo	68
Gráfica 13. Modularidad (Usuario Wisper 1)	82
Gráfica 14. Modularidad (Usuario Wisper 2)	83
Gráfica 15. Modularidad (Usuario Wisper 3)	84
Gráfica 16. Modularidad (Usuario Wisper 4)	85

Listado de Figuras

Figura 1. K3 y K5	19
Figura 2. K3 y K4	19
Figura 3. Ejemplo Comunidades.....	21

Figura 4. Ejemplo Modularidad	27
Figura 5. KMeans (Fuente: Univ. Carlos III de Madrid).....	30
Figura 6. Kernighan-Lin	34
Figura 7. Dendograma.....	36
Figura 8. Link Betweenness.....	38
Figura 9. Coeficiente de agrupamiento de enlaces.....	39
Figura 10. Problema del encadenamiento	41

Listado de Tablas

Tabla 1. Pruebas Calidad (Karate)	59
Tabla 2. Pruebas Calidad (Delfines)	60
Tabla 3. Pruebas Calidad (Fútbol)	61
Tabla 4. Pruebas Calidad (<i>Facebook</i> Francisco).....	62
Tabla 5. Pruebas Calidad (<i>Facebook</i> Aarón)	63
Tabla 6. Pruebas de Tiempo.....	67
Tabla 7. Permisos Redes Sociales	71
Tabla 8. Pruebas Calidad (Usuario Wisper 1).....	82
Tabla 9. Pruebas Calidad (Usuario Wisper 2).....	83
Tabla 10. Pruebas Calidad (Usuario Wisper 3).....	84
Tabla 11. Pruebas Calidad (Usuario Wisper 4).....	85

Listado de Diagramas

Diagrama 1. SuperClase CDAgorithm	52
Diagrama 2. Clases Algoritmos	53
Diagrama 3. Clases Medidas.....	54
Diagrama 4. Paralelismo MapReduce (Fuente: SlideShare).....	56
Diagrama 5. Secuencia Wisper	74
Diagrama 6. Clases Interfaz	78
Diagrama 7. Clases Modelo/Controlador.....	79

