San Cristóbal de La Laguna, 7 de Septiembre de 2019

Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en ingeniería informática

Algoritmos para la detección de comunidades en Redes

Algorithms for community detection in networks

Por: Miguel Jiménez Gomis

La presente memoria titulada:

"Algoritmos para la detección de comunidades en Redes"

ha sido realizada por D. **Miguel Jiménez Gomis**,

con N.I.F. 54063845-E.

Agradecimientos

A mi familia y amigos sin los cuales nunca podría haber terminado este trabajo, gracias por estar siempre ahí apoyándome, ayudándome y dándome ánimos para seguir adelante.

A todos los profesores que siempre están ahí para ayudar en todo lo que pueden y explicar con una paciencia inimaginable.

Licencia

Resumen

En este trabajo, se propone un nuevo algoritmo para la detección de comunidades en redes que permita realizar asignaciones rápidas a las diferentes comunidades existentes en el grafo, cuando se tratan con datos en tiempo real. El objetivo de este algoritmo es reducir la carga computacional de estos análisis y tener soluciones válidas aproximadas en el menor tiempo, haciendo posible la identificación de los datos existentes en el solapamiento de diferentes comunidades para tenerlos en cuenta y obtener mejores resultados.
Para comprobar la viabilidad del algoritmo propuesto se realizó una experimentación sobre data set reales, para poder compararlos de manera sencilla con la literatura; y con un conjunto de datos artificial, con el objetivo de que el análisis fuera más amplio. La experimentación realizada y análisis previos llevados a cabo en este proyecto establecen unas bases sólidas sobre las que seguir trabajando para completar y mejorar el algoritmo propuesto.

Palabras clave: clustering, comunidades, algoritmo, detección, big data.

Abstract

In this project, it is proposed a new algorithm for the detection of communities in networks that allows to make fast assignments to different communities in the graph, when data is in real time. The objective of this algorithm is to reduce the computational load of this analysis and provide approximate valid solutions in less time, making possible the identification of the data that exist in the intersection of the communities, to have those data points into account in order to obtain better results. To check the viability of the proposed algorithm, an experimentation has been done with real data to have an easy way to compare with the literature, and with an artificial dataset, in order to make the analysis broader.

The experimentation and previous analysis made in this project establish a solid base in order to keep working to complete and improve the proposed algorithm.

Keywords: clustering, communities, algorithm, detection, big data.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Con el auge del *big data* y la digitalización de nuestra sociedad se están recabando cantidades enormes de datos en muchos formatos, los cuales sin un debido análisis no sirven para mucho más que ocupar espacio en algún disco duro. Los problemas empiezan a surgir con la dificultad de procesar estas ingentes cantidades de datos a un ritmo que sea plausible, coste-efectivo por las computadoras actuales, y no solo en los grandes data centers y supercomputadores. Es por esto que los nuevos sets de datos cada vez más grandes requieren de análisis más exhaustivos [13] y por tanto nuevos métodos para resolver este problema tienen que ser presentados.

Una característica de los datos que se están recopilando actualmente es que tienden a no ser estáticos. Hace unos años cuando se recababa información se hacía mediante encuestas, registros, o procesos similares, lo cual ralentiza mucho el proceso de análisis y lo que se acababa tratando eran "snapshots" de esos datos en un cierto momento.

Los cambios de paradigma en la sociedad actual nos han llevado a que los datos sean casi instantáneos, cosa que solo aumentará con los años debido a la incorporación de nuevas tecnologías que lo facilitarán, como el 5G o la generalización de almacenar toda la información. Esto, junto con la existencia de una necesidad de inmediatez por parte de los interesados (analistas, usuarios,...) para que esos datos sean de utilidad lo antes posible, hace necesaria una gran labor de innovación en cuanto al análisis de toda esa información.

Otra característica destacable de los sets de datos actuales es que la mayoría se pueden representar en forma de grafo de relaciones. Estas estructuras de datos nos permiten extraer muchísima información relevante que de otras maneras no conseguimos. Por ejemplo, una de estas informaciones relevantes es la detección de comunidades en estos grafos, ya que al descubrir dichas comunidades se pueden inferir gustos, sacar atributos en común, definir perfiles, etc...

Actualmente existen numerosos estudios que tratan este tema desde diferentes ángulos, por ejemplo Yixin Chen *et al.* [16] que proponen el algoritmo *D-Stream* como sustituto al *K-means* para streams de datos en los que a los datos de entrada solo se pueden analizar 1 vez y en orden de llegada, o como Charu C. Aggarwal *et al.* [17] que proponen un algoritmo de clustering de grandes grafos empleando micro-clusters generados mediante técnicas compresión por hash para reducir el domino del problema. Si bien

estos algoritmos propuestos son cada vez más eficientes, siguen faltando metodologías para la asignación de clusters a datos individuales de una manera automática como una manera de pre-asignar los datos a diferentes clusters con precisión y sin necesidad de elevar el coste computacional.

Capítulo 2

2 Descripción del problema y objetivos

Para un problema cuyo dominio se puede representar en un grafo de relaciones entre los diferentes datos, se tiene un flujo de datos dinámicos que se van introduciendo en el sistema uno a uno, ya sea de forma regular o irregular. Para poder extraer la información más útil de estos datos, se necesita un algoritmo de detección de comunidades para encontrar relaciones con las que poder generalizar atributos entre los diferentes datos. Más concretamente, se requiere que estas comunidades permitan tener datos solapados para perder la menor cantidad de información importante posible.

Ya que los algoritmos para este tipo de problema no son muy comunes en la literatura, en este trabajo se propone un algoritmo para poder tratar este tipo de datos dinámicos dando una solución aproximada que suministre al usuario la mayor cantidad de información relevante con el mínimo de computación necesario.

Como se estableció en el anteproyecto de este trabajo de fin de grado, el plan de trabajo que se ha seguido es el siguiente:

1. Revisión bibliográfica de la literatura relativa a las técnicas y campos de aplicación de la detección de comunidades en redes complejas
2. Diseño e implementación de al menos una técnica de detección de comunidades solapadas en redes estáticas
3. Diseño de experimentos y comparativa con la literatura
4. Desarrollo de herramienta para representación gráfica de soluciones
5. Diseño e implementación de al menos una técnica de detección de comunidades solapadas en redes dinámicas

Capítulo 3

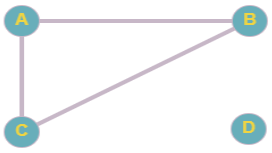
Antecedentes

3.1 Grafos

Para poder describir la detección de comunidades en grafos, primero hay que definir la estructura que entendemos por grafo. Desde un punto de vista general, entendemos por grafo a un conjunto de datos individuales (conocidos como nodos) conectados entre sí por medio de enlaces (aristas) que permiten representar las relaciones binarias que existen entre cada par de nodos relacionados entre sí.

De manera formal, un grafo es una estructura que está formada por dos conjuntos de elementos, el conjunto de nodos (V) y el conjunto de aristas (E). El conjunto de nodos es un conjunto no finito, en cambio el conjunto de aristas puede estar vacío, pero en cualquier caso, sus elementos estarán conformados por sub-conjuntos de pares del conjunto de nodos[1]. A los elementos del conjunto de nodos se les denomina nodo y a los elementos del conjunto de aristas arista, denotando por *"v" (del inglés vertex)* al número de elementos en el conjunto de nodos y por *"e"* (*del inglés edge*) al número de elementos en el conjunto aristas. Para cada par de elementos (x,y) ∈ E con x, y ∈ V decimos que esos dos elementos están relacionados o conectados entre sí y que, tanto x como y son adyacentes. Cada arista (x,y) es incidente en ambos nodos x e y. Un par de aristas que inciden en un mismo nodo se denominan adyacentes. Un nodo sin ninguna arista que incida en él, se conoce como nodo aislado.
Como ejemplo, sea el grafo G:

G = {V, E}
V = {A,B,C,D}
E = { {A,B},{B,C},{C,A} }



[1] Figura 1. Visualización del grafo usado como ejemplo en el apartado 3.1.

Se puede observar que v=4,e=3 y que D es un nodo aislado.

Una de las características que tienen los grafos es que sus datos pueden ser estáticos o dinámicos. Entendemos por datos estáticos aquella información que es inmutable en el tiempo y que, por tanto, su análisis no depende del momento en que éste se realice, por lo que independientemente del tiempo, los resultados de su análisis siempre serán similares. En cambio, los datos dinámicos son aquellos en los que estos sí dependen del factor tiempo, pues puede ser que se amplíen o se modifiquen mientras se realiza su análisis y tanto su almacenamiento como tratamiento se tienen que realizar teniendo siempre en cuenta el factor tiempo.

3.2 Comunidades en grafos

Entendemos por comunidad de un grafo G, al subgrafo C formado por los nodos que están fuertemente relacionados entre sí, con respecto a las uniones entre ellos, y no tanto con el resto del grafo G. Para poder describir de manera precisa qué es una comunidad, existen numerosas métricas que nos permiten cuantificar las relaciones existentes de una forma clara.
Para cada comunidad, podemos calcular el grado interno k_i^{int} y grado externo k_i^{ext} de cada nodo i respecto al subgrafo C con respecto al resto del grafo G. Estas medidas nos indican el número de aristas que conectan dos nodos del subgrafo G o un nodo del subgrafo G con otro fuera de este respectivamente. Si $k_i^{int} > 0$ y $k_i^{ext} = 0$, este nodo sólo tiene vecinos con elementos de la comunidad C y ninguno con el resto del grafo. Los nodos que tienen $k_i^{int} > 0$ y $k_i^{ext} > 0$ son de especial interés, pues tienen vecinos tanto en la comunidad como fuera de esta, conformando así la frontera que separa estos dos conjuntos. La incrustación ξ es el ratio entre el grado interno del nodo y el número total vecinos de este $\xi_i = k_i^{int} / K_i$ por lo que mientras mayor sea esa métrica mayor será la relación entre este nodo y su comunidad. Esto nos permite detectar qué nodos son más representativos de cada comunidad pudiendo extraer de ellos información importante. El parámetro de mezcla μ_i es el ratio entre el grado externo de un nodo y el número total de vecinos de este. $\mu_i = k_i^{ext} / k_i$. Por definición $\mu_i = 1 - \xi_i$.

En función de estas variables podemos definir algunas de las métricas más usadas en la detección de comunidades:

- Grado interno k_C^{int} : es la suma de todos los aristas internos de la comunidad C. $k_C^{int} = \sum ij \in_C A_{ij}$ Siendo A_{ij} la matriz de adyacencia del grafo G.
- Grado interno promedio $k_c^{avg\text{-}int}$ es el promedio de vértices de C considerando solo las aristas internas $k_c^{avg\text{-}int} = k_C^{int} / n_C$.

- Densidad de aristas internas es el ratio entre el número de aristas internas de C y el número de aristas internas totales.

$$\delta^{int} C = \frac{k^{int} C}{nC(nC - 1)}$$

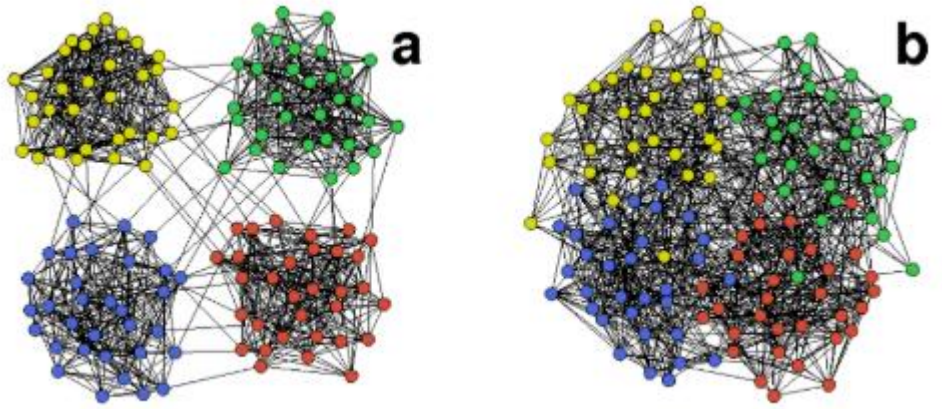
- Grado externo k_C^{int} : es la suma de todas los vértices externas de la comunidad C. $k_C^{ext} = \sum i \in_C j \notin C A_{ij}$ Siendo A_{ij} la matriz de adyacencia del grafo G.
- Grado interno promedio $k_c^{avg\text{-}ext}$ es el promedio de vértices de C considerando solo las aristas externas $k_c^{avg\text{-}int} = k_C^{int} / n_C$.
- Densidad de aristas internas es el ratio entre el número de aristas externas de C y el número de aristas internas totales.

$$\delta^{ect} C = \frac{k^{ext} C}{nC(nC - 1)}$$

Una vez tenemos el concepto de comunidad, podemos hablar de los dos tipos de comunidades que nos podremos encontrar, y estas son las comunidades fuertes o débiles.

Se entiende por comunidad fuerte aquella que sus nodos tienen mayor probabilidad de ser adyacentes con cualquier otro nodo de la propia comunidad antes que con el resto del grafo.

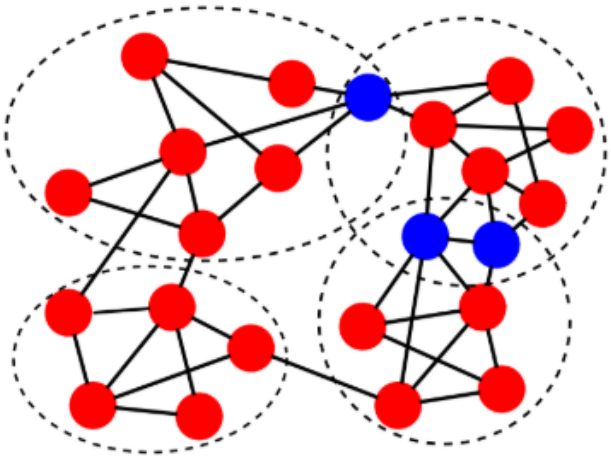
Por el contrario, se entiende por comunidad débil aquella en la cual la probabilidad media de cada nodo de tener aristas con la propia comunidad, excede a la probabilidad de tener arista con cualquier nodo de otra comunidad. La *Figura 2* se pueden observar un ejemplo de una comunidad fuerte (a) y otra débil (b).



[2]

3.3 Comunidades solapadas

En los grafos, las distintas comunidades, con excepción de las comunidades fuertes que están claramente diferenciadas unas de otras, tienden a estar próximas entre sí y sus fronteras tienden a solapar, como se puede ver en la *Figura 3*. Muchos algoritmos, conocidos como *clustering duro (hard clustering)* asignan estos elementos en las fronteras de los clusters a un elemento o a otro sin diferenciar si estos elementos podrían formar parte de ambas comunidades.



[3]

La definición de comunidad solapada no es una formulación matemática precisa, sino más bien un evento que aparece en los grafos y se intenta cuantificar. Es por esto que para poder diferenciar claramente qué elementos se encuentran o no dentro de una

[2] *Figura 2.* Ejemplo de comunidades fuertes(a) y comunidades débiles (b). Imagen sacada de [2].
[3] *Figura 3.* Ejemplo del solapamiento existente en las fronteras de comunidades próximas entre sí. Imagen sacada de [2].

comunidad, se deben definir unas nuevas métricas que sirvan para describir dichas comunidades. Aquí surge el problema de que no hay una medida única por la cual se puedan detectar estos solapamientos y por tanto múltiples métricas se tienen que usar de manera combinada para tratar de inferir si un nodo está solapado o no.

- *Modularidad*: La modularidad es una medida de estructura de la red que mide la calidad de la detección de comunidades. Fue propuesta por primera vez por Girvan y Newman [3]. Redes con una alta modularidad indican que están fuertemente conectadas dentro de cada comunidad y poco conectadas entre diferentes comunidades. M =Σ (i) [(ls/L) - (ds/2L)^2] iterando i sobre las distintas comunidades, los números de enlaces con ambos extremos en la misma comunidad, de aquellos enlaces con extremos en diferentes comunidades y L el número total de enlaces en el grafo.

- *Average degree of the community:* K^{avgC} [4]: AD(c)=2*(|E(c)|/|c|), donde E(c) es el número de links internos de la comunidad (ambos nodos están en la misma comunidad (c)) y |c| es el numero de vertices en c. Si el añadir un nodo hace aumentar el AD, se asume que contribuye a dicha comunidad.

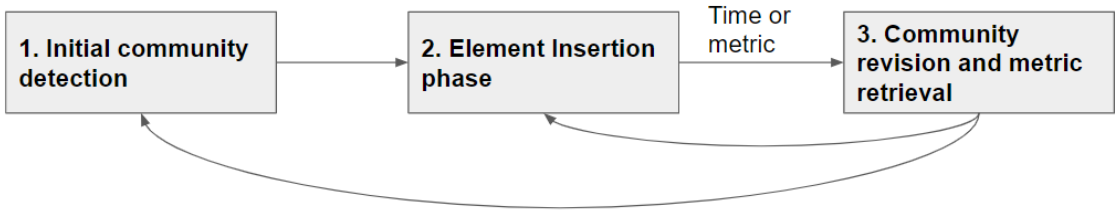
- *Equivalencia estructural:* La estructura local de un vértice se puede definir como Γ(v) = {w e V|(v,w) e E} U {v}. entonces se define la similaridad estructural basada en la intuición de que a mayor número de amigos tengan dos personas en común, mayor será la probabilidad de que sean amigos. El método más sencillo para definir la similaridad estructural es usando el número de vecinos en común. El rango de similaridad estructural sería incierto y cambiaría entre redes, por eso se normaliza el número común de vecinos por la media geométrica de las 2 vecindades

$$\sigma(u,v)=(|\Gamma(u) \cap \Gamma(v)|)/\sqrt{(|\Gamma(u)| * |\Gamma(v)|)}$$

Capítulo 4

Algoritmo propuesto

Para poder resolver el problema de la detección de comunidades sobre grafos con datos dinámicos, se propone el siguiente algoritmo como una opción sencilla de implementar para obtener resultados válidos con el mínimo esfuerzo computacional posible.
Este algoritmo consiste en tres etapas diferenciadas las unas de las otras.



[4]

4.1 Primera etapa. Detección de comunidades iniciales.

En esta primera fase, sobre un conjunto de datos de tiempo real recabados pero no analizados, se ejecutará un algoritmo clásico de detección de comunidades solapadas de la literatura existente, como por ejemplo CPM [5], CONGA [6], Fuzzy Clustering [7], o LED [8] (este último es el que se ha seleccionado para usarse en la experiencia computacional y será explicado detalladamente en dicha sección). Con la ejecución de este algoritmo se conseguirá generar un conjunto de comunidades iniciales, al que llamaremos I_C, a partir de las cuales el proceso trabajará para asignar los diferentes elementos que se vayan adhiriendo al grafo principal. Este paso es el que más errores puede generar en la ejecución y por tanto es crucial que el algoritmo ejecutado sea eficaz y eficiente en la detección de estas comunidades pues de ello dependerá que el proceso general sea exitoso. Es por esto que una experimentación en profundidad es requerida para la determinación de este algoritmo, pero por motivos del alcance de este proyecto, ese paso se ha tenido que omitir. En la sección de experimentación se detalla cómo se ha realizado la experimentación para paliar los efectos de estos errores así como las posibles mejoras en el mismo.

[4] *Figura 4*, diagrama de ejecución de las diferentes fases del algoritmo propuesto.

4.2 Segunda etapa. Inserción de elementos en las comunidades iniciales.

En esta fase, los diferentes nodos que se vayan insertando en el grafo serán analizados por medio de diferentes métricas y asignados a una o varias comunidades, consiguiendo así mantener el solapamiento.

Este paso es la parte principal de la metodología, en él se irán asignando todos los datos que vayan entrando, aunque, es esperable que a medida que más datos sean introducidos en sus respectivas comunidades éstas empeoren drásticamente. Es por eso que cada cierto tiempo, marcado por una métrica de tiempo o de calidad de las comunidades generadas, se pasará a la fase 3 en la que todas las comunidades serán evaluadas y se decidirá como seguir procediendo con la metodología.

El punto más crítico en esta fase es la selección de las métricas a emplear para esta asignación inmediata. Dependiendo de cómo de acertada sea esta métrica, el elemento estará, o no, bien asignado a la comunidad que le corresponda y por tanto la metodología será efectiva. La selección de dicha métrica corresponderá principalmente al dominio del problema a tratar. En el capítulo 5 de este trabajo, Experiencia computacional, será probada las métricas de Modularidad.

La implementación de la inserción por modularidad es relativamente sencilla. Esta consiste en insertar el elemento en cada una de las diferentes comunidades y calcular la modularidad de esta. Una vez realizado esto con todas las comunidades, el elemento se insertará en aquellas con las que supere cierto margen de error para así poder simular una inserción de elementos solapados. En la *Tabla 1* se puede ver el pseudocódigo de esta implementación:

Tabla 1.- Pseudocódigo de la inserción por medio de Modularidad.

```
Require graph G, α, initialCommunities, error (%)
1.  maxModularity = 0;
2. insertions = [];
3. For each community  in initialCommunities:
4.calculate modularity of data;
5. If datamodularity >= max Modularity * (error/100) :
6. insertions.add community;
7. End Loop;
8. For each community in insertions:
9 add data to community;
10. End Loop;
```

Otra posible métrica sería la similaridad estructural promedio de cada comunidad. Esta implementación consiste en que para cada dato a insertar en el grafo, y para cada comunidad se calcula su similaridad estructural con todos los elementos de dicha comunidad y se promedian. Después, al igual que con la Modularidad, el dato será insertado en aquella comunidades cuya similaridad estructural promedio sea superior a un margen de error dado como input. Este margen de error se calcula utilizando la similaridad estructural promedio máxima. En la Tabla 2 se puede ver el pseudocódigo de esta implementación.

Tabla 2.- Pseudocódigo de la inserción por medio de similaridad estructural promedio

```
Require graph G, α, initialCommunities, error (%)
1.  maxAverageStructuralSimilarity = 0;
2. insertions = [];
3. For each community  in initialCommunities:
4.calculate Average Structural Similarity  of data;
5. If datamaxAverageStructuralSimilarity  >= maxAverageStructuralSimilarity  * (error/100) :
6. insertions.add community;
7. End Loop;
8. For each community in insertions:
9 add data to community;
10. End Loop;
```

Aunque en este trabajo solo se implementen estas dos métricas por motivos de alcance y tiempo, la experimentación solo se realiza usando la inserción por modularidad promedio.Un estudio en profundidad se puede realizar de las muchas otras métricas existentes en el campo del análisis de grafos, lo cual abre muchas posibilidades a futuras investigaciones continuando por esta rama.

4.3 Tercera etapa Revisión de la calidad de las comunidades.

Cada cierto tiempo de ejecución, la metodología pasará a la etapa 3. En esta etapa, se evaluarán las calidades de las comunidades y se tomará la decisión de seguir insertando datos, es decir, continuar la etapa 2, o detener esta inserción y pasar a la etapa 1. Este paso es debido a que al estar insertando valores directamente en las comunidades presumiblemente empeorará las calidades de éstas frente a su estado inicial. Es por ello que cuando este deterioro de la comunidad pase un umbral de

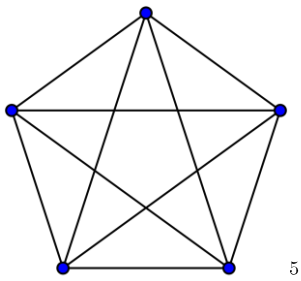
calidad (o un tiempo fijo establecido) las comunidades se desharán y se volverá a pasar a la etapa 1 con todos los datos recopilados hasta el momento. Esto generará nuevas comunidades iniciales o continuará con las mismas, pero de una manera en la que probablemente mejore los resultados de las inserciones por sí mismas. Con este paso se consigue mantener una cierta calidad general evitando computar algoritmos más pesados constantemente, aunque siga siendo necesario ejecutarlo múltiples veces. En función de cómo de acertadas sean las métricas o heurísticas que se usen en el proceso de asignación de la etapa 2, estos algoritmos tendrán que ser ejecutados menos veces y por tanto mejorará el rendimiento general del proceso. La principal métrica usada en la literatura para el análisis es la modularidad de las comunidades,y es la que en un futuro se debería usar en una experimentación exhaustiva de esta tercera fase.

Capítulo 5

Experiencia computacional

5.1 Diseño del experimento

La mejor manera de obtener unos buenos resultados a la hora de analizar el rendimiento del algoritmo propuesto es la creación de un dominio sintético en el que todos los datos sean trazables de antemano y que las comparaciones con los resultados generados nos indiquen de manera precisa cómo está operando el algoritmo
Para la sintetización de este dominio artificial se va a implementar diferentes comunidades utilizando la estructura de n-clique .
Un n-clique es definido en [9] de la siguiente manera; "Definimos un n-clique de un grafo G como un subgrafo de F inducido por un set de puntos V , asociado con el subgrafo maximal completo de la potencia del grafo G_n. Esto es, cada subgrafo maximal completa de la potencia del grafo G_n tiene un set de puntos (o individuos) asociados con él; este set de puntos junto al set de líneas (o relaciones) conectandolos en el grafo original forman un subgrafo al que llamamo n-clique.
Ya que cada par de puntos en un n-clique es adyacente en la potencia del grafo G_n, la distancia entre dos puntos en el grafo original es menor o igual que n." La *Figura 5* muestra un ejemplo tradicional de clique de tamaño n = 5. Las particularidades de los n-cliques los convierten en comunidades perfectas ya que, cuando se encuentran aislados, tienen una Modularidad muy alta, lo cual nos indica que esos nodos están fuertemente conectados entre sí, y débilmente conectados con otras partes del grafo.



[5]

[5] Figura 5. Clique de tamaño n = 5. Imagen con licencia de Dominio público. Sacada de Wikipedia (https://es.wikipedia.org/wiki/Clique) (Visitado ultima vez el 20/07/2019)

La ventaja que nos proporcionan los cliques es, entre otras, la capacidad de agrandar su tamaño sin que pierda sus propiedades, lo cual es ideal para un entorno de experimentación.

Una vez se generen los diferentes cliques como comunidades iniciales, se establecerán enlaces aleatorios (los cuales serán registrados) entre los diferentes cliques, con esto se instanciará un grafo mucho más grande y complejo pero con unas comunidades muy bien definidas. Con esto obtenemos la posibilidad de crear un tamaño de experimentación variable con el cual probar de una manera sencilla y fácil de revisar la metodología propuesta a diferentes escalas y densidades de conexión entre comunidades.

Tras esto, un número n de elementos serán sacados del grafo y pasarán a ser nuestro flujo de datos en tiempo real que se irán analizando e introduciendo de nuevo a sus comunidades en la segunda fase de la metodología.
Es esperable que, a mayor número de comunidades iniciales y mayor tamaño de estas, el algoritmo empeore en general ya que uno de los factores que más influyen en este tipo de análisis es el tamaño. Con esto también podemos fijar un tercer parámetro en la experimentación, pues podemos comprobar como afecta la cantidad de datos introducidos en el proceso en general.
La parte más variable de esta metodología es sin duda alguna la generación de la comunidad inicial, pues esto influirá no solo en el tiempo de ejecución sino también en la calidad del proceso en general. Es por esto que en esta fase de experimentación sintética, el paso inicial se hará asumiendo un algoritmo que funcione a la perfección en la detección de estas comunidades haciendo uso del conocimiento previo que tenemos de la estructura de estas comunidades.

La segunda parte de esta experimentación consistirá en el análisis de dos casos de estudios comunes en la literatura, siendo estos los datasets de “Karate club” [10] y “College football” [11]. Estos datasets consisten en grafos representando diferentes relaciones en sus respectivos dominios y, al estar ampliamente estudiados por la literatura son un perfecto banco de pruebas para comparar los resultados del algoritmo que propongo en este trabajo con casos reales ya documentados.

5.2 Algoritmo LED (Loop edge detection)

Como ya se ha comentado con anterioridad, se ha seleccionado el algoritmo Loop edge detection [8] como el algoritmo a ejecutar en la primera fase de este algoritmo como pieza generadora de las comunidades iniciales. Este algoritmo fue elegido por su facilidad de implementación y su rapidez al ejecutar.

El funcionamiento de este algoritmo es el siguiente:

Dado un grafo G se calcula la equivalencia estructural para todas las aristas pertenecientes a dicho grafo. Todas aquellas aristas con un valor inferior a un parámetro de entrada αson eliminadas del grafo y se almacenan en una lista en el orden que van siendo eliminadas. Este proceso se tiene que repetir hasta que no se puedan eliminar más aristas.

Una vez se tenga el grafo con las aristas con equivalencia estructural inferior a α eliminadas, se marcan como comunidades iniciales aquellas componentes conexas del grafo.

Ahora, para cada arista eliminada en la lista, se van insertando de manera inversa a como fueron eliminadas en el grafo. Si los nodos de la arista están en comunidades diferentes, se juzga si estos nodos están solapados por medio del grado promedio de la comunidad. Por último, los posibles nodos que hayan quedado aislados son insertados en la comunidad con la que tengan mayor similaridad estructural.

***Tabla* 3.- Pseudocódigo del algoritmo LED. Extraído de [8]**

Algorithm 1. LED algorithm.
Require: Graph G, α
Ensure: Clusters Ci; where i∈ (1; 2; ...; k).
1: curG=G; //pointer to current graph
2: edgeListi; //store the ith deleted edges
3: graphListti;//store the ith graph
//Step1 and Step2: Calculating Structural Similarity and
Deleting edges less than α
4: repeat
5: for each edge in curG do
6: Calculate Structural Similarity;
7: end for
8: push curG into graphList;
9: copy curG to G';
10: if w(e) < α then
11: curG = curG - {e};

12: edgeListi = edgeListi + (e);
13: end if
14: until edgeListi is not empty
15: label each inter-connected-components as an initial community Ci;
//Step3: Identify Overlapping Vertices
16: for each edgeListi in reverse order do
17: if two vertices of the edge in different initial communities then
18: judge the vertices of the edge is overlapping or not using AD(c) (formula(4));
19: end if
20: end for
//extend initial communities
21: attach the isolated vertices to the one initial community with highest Structural Similarity; return Clusters;

Para facilitar la implementación de este algoritmo se realizó un pequeño cambio a la estructura de este. La primera y segunda fase que en el paper indican que se realice de forma iterativa, se optó por un acercamiento recursivo, con la finalidad de simplificar el bucle lo máximo posible. El pseudocódigo de mi implementación de LED se queda de la siguiente manera:

***Tabla 4.*- Modificación de LED. El código de esta tabla sustituye las líneas desde la 4 hasta la 14 del pseudocódigo del algoritmo original.**

4: def recursive (graph, α):
5: Calculate Structural Similarity;
6. Delete edges;
7. Check if G.inter-connected-components == 1:
8: Mark G as initial community;
9. Else:
10: For each inter-connected-components in G:
11. recursive(inter-connected-components, α);
12. end IF;

Este método recursivo funciona de tal manera que una vez se han eliminado las diferentes aristas con menor equivalencia estructural, se calculan las diferentes componentes conexas del grafo. En caso de que solo exista una componente, esta se marca como comunidad inicial y se acaba la recursividad. En caso de que hayan múltiples componentes conexas, se toman cada una de estas como grafo de entrada para el método recursivo manteniendo el mismo valor α del input.

5.3 Implementación de la experiencia computacional

La implementación de la experiencia computacional se ha realizado en el lenguaje de programación Java. Este lenguaje se escogió por su potencia a la hora de realizar cálculos más complejos, cosa que otros lenguajes como por ejemplo Python no permiten pues son mucho menos eficientes a la hora de ejecutar las instrucciones. Otro de los factores que fue clave para la selección de Java como lenguaje de desarrollo fue la facilidad de implementación en diferentes entornos, su compatibilidad entre sistemas operativos y la disponibilidad de numerosas librerías que permiten que el desarrollo se centre sólo en la parte técnica sin tener que preocuparse de otros factores que se pueden automatizar. En el caso concreto de este proyecto se ha utilizado la librería JGraphT [14] como medio para la definición de la estructuras de datos de los grafos, pues esta librería permite instanciarlos de una manera muy sencilla sin tener que programar la estructura de datos desde cero y trae consigo numerosas funcionalidades que facilitan mucho el trabajo sobre grafos, como son métodos para obtener adyacencias y componentes conexas, algoritmos de recorridos de grafos,...

Todo el código de la implementación de este proyecto se puede encontrar debidamente documentado en el siguiente repositorio de GitHub: https://github.com/MiguelJG/TFG. Para su ejecución es necesario que se tenga instalada una máquina virtual de Java, y recomiendo la utilización del IDE Eclipse, pues este trabajo se ha estructurado como un proyecto para este entorno de desarrollo. No es necesario descargar las diferentes librerías externas pues estas se encuentran ya en el repositorio del proyecto.

5.4 Herramienta para representación gráfica de soluciones

La manera más representativa de reflejar fielmente comunidades en un grafo es dibujando sobre un plano los nodos y aristas de este y coloreando del color adecuado cada nodo, siendo este color el que indique la comunidad habiendo preasignado colores a estas, como se puede ver por ejemplo en la Figura 2.

Implementar una herramienta desde cero para esta funcionalidad habría conllevado un gran esfuerzo que, por razones de tiempo disponible, se decidió no hacer y se buscaron alternativas. La primera librería que se intentó utilizar fue JGraph[18] pero debido a que su uso no era específico para la visualización de grafos , pues esta está más enfocada a diagramas de componentes para un entorno web, y que los repositorios para java están faltos de documentación, no se utilizó.
Tras una búsqueda en profundidad entre las diferentes librerías que servirían para el representar las comunidades se decidió utilizar Graph Stream [19] como librería para representar los grafos. Sin embargo, debido a falta de tiempo, la adaptación de esta librería a las estructuras de grafos generadas por JGraphT se quedó en un estado inestable en la que no mostraba las gráficas como se esperaba por lo que se ha decidido no incluir los resultados en la entrega final del proyecto.

5.5 Metodología de desarrollo

La metodología de desarrollo que se ha seguido para esta implementación es llamada "método Kanban" que consiste en definir tareas con prioridades e ir realizándolas y moviéndolas entre los diferentes estados que son "Propuestas", "Preparadas", "En desarrollo" , "Revisión" y "Completas". Esta metodología permite llevar un control del flujo de trabajo a lo largo de todo el proyecto haciendo la toma de decisiones más flexible y asegurando que ninguna tarea se pierde por el camino.

5.6 Resultados de la experiencia computacional

El primer experimento que se realizó fue con datos reales. Para tener una referencia clara con la literatura se usaron los datasets de *Karate club* y *College football*.
En la Tabla 5 se puede observar el output generado por el programa para dos de los casos del estudio. En esta tabla se puede ver como la modularidad inicial, que es la dada por el algoritmo LED, va mejorando en función de las nuevas inserciones de datos que se van realizando por el algoritmo de la segunda fase para la primera métrica (con la modularidad).

Tabla 5.- Ejemplo de output

Karate: 50%	[Init:0.08584043421635067	0:0.09768049410826109	1:0.09399468539381972	2:0.10341814395936527	3:0.09965771865291029
	4:0.11043152607484653	5:0.10651111309137723	6:0.0976627260230579	7:0.09292490191673984	8:0.10927017858449642
	9:0.10583776684667907	10:0.12545129759045792	11:0.1257536841177242	12:0.13799999999999998	13:0.1417270012496024
	14:0.1640586899713655	15:0.2286394456622278	16:0.33645316800701813]		
Football: 90%	[Init:0.05455563624354258	0:0.05777997107911524	1:0.059904	2:0.06100027142427686	3:0.0639940164642304
	4:0.0662735500056825	5:0.06967257664383456	6:0.07349992822276138	7:0.0745844747787063	8:0.0807056482930449
	9:0.0458209346501329	10:0.05038561318926301	11:0.06295292548589213]		

En la Tabla 6, se observan los resultados completos de la experimentación. En esta tabla se puede comparar la diferente modularidad inicial y final en función del porcentaje de nodos que se encontraban en el grafo a la hora de establecer las comunidades iniciales. Como se ha comentado anteriormente, el porcentaje de nodos iniciales influye mucho a la hora de establecer unas buenas comunidades iniciales. Esto se puede observar claramente en la tabla que, a mayor número de nodos en el grafo cuando se detectan estas comunidades, mayor es la modularidad.

Por otra parte, si observamos la modularidad final en estas tablas, se ve claramente como la inserción por medio de la modularidad consigue mejorar los resultados finales con respecto al inicial en todos los casos.

Si comparamos los resultados de esta experimentación con los resultados obtenidos en la literatura, podemos ver como los algoritmos que calculan las comunidades sobre todo el conjunto de datos tienen unos resultados algo mejores que los obtenidos con el algoritmo propuesto en este trabajo pues tienen más información con la que trabajar y de la que extraer relaciones.

Para el dataset de Karate Club, con el algoritmo Girvan–Newman[20] (GN de aquí en adelante) la modularidad del grafo es de 0.4012 (Dato obtenido de [8]), con el algoritmo LED ejecutado en solitario esta es de 0.4155 mientras que el promedio de los resultados obtenidos en la experimentación del algoritmo propuesto en este trabajo ha sido de 0.3384.

Por otra parte, para el dataset de College Football se tiene que con el algoritmo GN la modularidad del grafo tras detectar las comunidades es de 0.5995 (Dato obtenido de [8]) mientras que con el algoritmo LED es de 0.5995 y con el algoritmo propuesto en este trabajo el promedio de la experimentación es de 0.4383.

Tabla 6.- Ejemplo con datos reales

Karate club dataset			
n nodos	34		
% de nodos iniciales	Mod inicial	Numero de insercion es	Mod final
10	0.01776008983786565	30	0.3516153268219384
30	0.062160314432529784	23	0.3462234462175538
50	0.08584043421635067	17	0.33645316800701813
70	0.10360052405421631	10	0.3367305503318155
90	0.1332006737839924	3	0.32143477176559476
College footbal			
n nodos	115		
% de nodos iniciales	Mod inicial	Numero de insercion es	Mod final
10	0.002899091781057818	119	0.8515159050521792
30	0.016472112392373967	93	0.6236254483254857
50	0.030835794398524064	66	0.428625145345822
70	0.044408815009840216	37	0.2251049497997793
90	0.05455563624354258	12	0.06295292548589213

El segundo experimento fue sobre los grafos artificiales generados, como se explicó en el apartado anterior, por medio del uso de n-cliques. Para hacer la experiencia computacional completa este experimento se realizó ejecutando para cada número de clúster con un tamaño incremental de nodos por clúster variando desde 5 hasta 70 nodos por clúster en incrementos de 5 para cada porcentaje inicial de nodos. Con todos los datos de la experimentación se han construido dos matrices que se pueden ver en las tablas 7 y 8. En la tabla 7 está reflejado el incremento de la modularidad tras la inserción de todos los nodos extraídos del grafo. Como era de esperar, a menor tamaño del grafo, mayor este incremento en la mejora.

Tabla 7.- Matriz del incremento de modularidad promedio con datos artificiales

		% inicial de nodos							
		10	20	30	40	50	60	70	80
Numero de clusters	2	0,35122	0,27268	0,11063	0,08844	0,05864	0,0491	0,03337	0,02377
	3	0,37264	0,2724	0,08058	0,05786	0,03888	0,03228	0,02201	0,01562
	4	0,34014	0,19315	0,05661	0,04111	0,02842	0,02426	0,01638	0,01166
	5	0,30716	0,17899	0,04122	0,03235	0,02257	0,01876	0,01275	0,00911
	6	0,22321	0,15336	0,02378	0,02461	0,01806	0,01495	0,0102	0,00728
	7	0,18086	0,05113	0,01982	0,02199	0,01487	0,01232	0,0084	0,00599
	8	0,15657	0,06306	0,01615	0,01613	0,01253	0,01038	0,0071	0,00505
	9	0,15611		0,01486	0,01434	0,01076	0,0089	0,00611	0,00435

En la tabla 8 está reflejada solamente la modularidad inicial de estas comunidades iniciales. Esta tabla refleja claramente cómo a menor número de comunidades la modularidad aumenta, tal y como era de esperar pues hay menos conexiones entre los nodos internos de cada comunidad con otras comunidades. Se puede observar en la

tabla que a menor número de comunidades por clúster y a mayor número de nodos en este grafo, la modularidad inicial en el grafo mejora. En cambio, cuando el grafo empieza a ser muy grande, pero se tienen pocos nodos de este, la modularidad inicial empeora.

Tabla 8.- Matriz de modularidad inicial

		% inicial de nodos							
		10	20	30	40	50	60	70	80
Numero de clusters	2	0,10712	0,11704	0,12781	0,13018	0,1341	0,13482	0,13624	0,13656
	3	0,05972	0,06505	0,07232	0,07607	0,0815	0,08375	0,087	0,08854
	4	0,03897	0,04251	0,04768	0,0512	0,05607	0,06004	0,0637	0,06547
	5	0,0284	0,03118	0,03532	0,03836	0,04274	0,04492	0,04822	0,04993
	6	0,02152	0,0237	0,02697	0,02956	0,03326	0,03525	0,03816	0,03972
	7	0,017	0,01877	0,02143	0,02365	0,02682	0,02858	0,03117	0,03259
	8	0,01386	0,01533	0,01756	0,01948	0,02223	0,02377	0,02606	0,02737
	9	0,01158	0,01282	0,01472	0,01639	0,0188	0,02018	0,02221	0,0234

Si se comparan estas dos matrices se puede ver claramente cómo este método de inserción funciona mejor mientras mayor número de inserciones tenga que hacer y mayor modularidad inicial tenga, lo que deja claro que un principal factor de sesgo para este algoritmo es la primera etapa de generación de comunidades iniciales y que mientras mejor sea ésta, mejor será la modularidad en las inserciones de la segunda fase del algoritmo.

Capítulo 6

Conclusiones y Líneas futuras

6.1 Conclusiones finales

La detección de comunidades en redes es una tarea computacionalmente muy exigente y el no tener algoritmos capaces de analizar datos en tiempo real de una manera sencilla y cómoda dificulta notablemente muchas investigaciones. En este trabajo se ha propuesto un algoritmo para intentar solventar este problema y se ha realizado una experimentación para comprobar la viabilidad del mismo.

Como se pudo observar en el capítulo *5.5 Resultados de la experiencia computacional* el algoritmo propuesto genera unos resultados que, como era de esperar, no son tan buenos como los generados por otros algoritmos, pero que para la tarea de asignación provisional son prometedores.

Tras realizar la experimentación se puede concluir que la principal fuente de error para los resultados intermedios es trabajar sobre un set de comunidades iniciales que no sea bueno. Este paso influye tanto, que para mejorar los resultados finales se debería continuar la investigación en la detección de comunidades con pocos nodos y así mejorar el funcionamiento en común del algoritmo.

Si bien los resultados obtenidos no son de la misma calidad que otros algoritmos ya disponibles, una primera versión del algoritmo propuesto en este trabajo se podría implementar en proyectos de prueba para ver su rendimiento en casos reales de uso, pues el desarrollo del código fue orientado a objetos y módulos con el fin de, en un futuro, facilitar la generación de una API que de servicio con este algoritmo.

Con el objetivo de mejorar este algoritmo y que en algún momento llegue a ser un producto viable con un impacto positivo, se debe continuar con una investigación más a fondo de las diferentes fases e implementar una experimentación más exhaustiva.

Este TFG me ha brindado una oportunidad perfecta para aprender sobre el análisis de redes y las muchas ramas que hay en esta fascinante disciplina, poniendo a prueba todos los conocimientos que he adquirido a lo largo de estos 4 años de carrera.

6.2 Líneas futuras

La realización de este trabajo a dejado una infinidad de posibles líneas futuras sobre las que seguir trabajando para desarrollar este algoritmo aún más en profundidad, destacando dos grupos principales:

- Profundizar en el análisis y la investigación de este algoritmo para mejorarlo. Debido al tamaño de este trabajo hay muchas líneas de investigación que no se han podido cerrar. De entre estas destacan la búsqueda de algoritmos de clustering más eficaces para la detección de comunidades iniciales de mejor calidad, desarrollar y analizar nuevas métricas y modelos para la segunda fase con diversas técnicas como puedan ser Machine Learning, modelos probabilísticos... todo lo que mejore los resultados del algoritmo en su totalidad, ampliando siempre la experimentación y mejorando los análisis.
- Implementar una versión usable del algoritmo. Este algoritmo se ha desarrollado con la idea de ser usado para experimentación y ver si podría llegar a ser útil o no. Ahora que se tienen unos resultados prometedores se podría trabajar en su implementación como librería para algún framework ya establecido, como podría ser *Weka*[15] , implementarlo como una API propia, o hacer una aplicación completa con el objetivo de que se puedan resolver problemas en dominios reales.

Capítulo 7

Summary and Conclusions

7.1 Final conclusions

The detection of communities in networks is a computationally demanding task and not having algorithms capable of analyzing data in real time in a simple and convenient way makes many investigations very difficult. In this work an algorithm has been proposed to try to solve this problem and an experimentation has been carried out to verify its viability.

As seen in chapter *5.5 Results of the computational experience*, the proposed algorithm generates results that, as expected, are not as good as those generated by other algorithms, but which are promising for the provisional assignment task.

After conducting the experiment, it can be concluded that the main source of error for the intermediate results is to work on a set of initial communities that is not good. This step influences so much that, in order to improve the results, research should continue in the detection of communities with few nodes and thus improve the common functioning of the algorithm.

Although the obtained results are not of the same quality as other algorithms already available, a first version of the proposed algorithm in this work could be implemented in test projects to see its performance in real use cases, since the development of the code was oriented to objects and modules in order to, in the future, facilitate the generation of an API that serves this algorithm.

With the aim of improving this algorithm and that at some point it becomes a viable product with a positive impact, a more thorough investigation of the different phases must be continued, and a more thorough experimentation must be implemented.

This TFG has given me the perfect opportunity to learn about network analysis and the many branches that are in this fascinating discipline, and thus, testing all the knowledge that I have acquired throughout these 4 years of career.

7.2 Future Work

The realization of this work has left an infinity of possible future lines on which to continue working to develop this algorithm even more in depth, highlighting two main groups:

- Deepen the analysis and investigation of this algorithm to improve it. Due to the size of this work there are many lines of research that could not be closed. Among these are the search for more efficient clustering algorithms for the detection of better quality initial communities, develop and analyze new metrics and models for the second phase with various techniques such as Machine Learning, probabilistic models, ... everything that improves the results of the algorithm in its entirety, always expanding the experimentation and improving the analysis.
- Implement a usable version of the algorithm. This algorithm has been developed with the idea of being used for experimentation and see if it could be useful or not. Now that there are promising results, it could be possible to work on its implementation as a library for an already established framework, such as Weka [15], implement it as your own API, or make a complete application in order to solve problems in real domains

Capítulo 8

Presupuesto

8.1 Coste del proyecto

Para la realización de este proyecto han sido necesarias alrededor de 350 horas de trabajo efectivo. Siendo el trabajo un puesto para alguien con una formación superior universitaria, ciñéndonos al promedio de 10€/hora [21] de sueldo para profesionales de la investigación y asumiendo que los costes de equipamiento y material son resueltos por la universidad sin coste añadido para este proyecto en concreto, la realización del proyecto resulta en un total de 3500€. Estos costes se pueden ver desglosados en la tabla 9.

Tabla 9.- Costes del proyecto

Número de horas trabajadas	350
Coste por hora	10 €
Total	3.500 €

Bibliografía

[1] Trudeau, Richard J. Introduction to Graph Theory. New York: Dover, 1993. Print.
[2] Santo Fortunato. Community detection in networks: A user guide
[3] M. Girvan, M.E. Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. 99 (12) (2002) 7821–7826.
[4] Y. Cai, C. Shi, Y. Dong, Q. Ke, B. Wu, A novel genetic algorithm for overlapping
community detection, in: Advanced Data Mining and Applications, Springer, Beijing, China, 2011, pp. 97–108.
[5] H. Sun, J. Huang, X. Zhang, et al., IncOrder: incremental density-based community detection in dynamic networks, Knowl.-Based Syst. 72 (2014) 1–12.
[6] M.E. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113.
[7] S. Zhang, R.-S. Wang, X.-S. Zhang, Identification of overlapping community structure in complex networks using fuzzy c-means clustering, Phys. A: Stat. Mech. Appl. 374 (1) (2007) 483–490.
[8] T. Ma, Y. Wang, M. Tang, J. Cao, Y. Tian
Abdullah al-dhelaan, mznah al-rodhaan, LED: a fast overlapping communities detection algorithm based on structural clustering
Neurocomputing, 207 (2016), pp. 488-500
[9] Alba, Richard D. (1973), "A graph-theoretic definition of a sociometric clique" (PDF), Journal of Mathematical Sociology, 3 (1): 113–126, doi:10.1080/0022250X.1973.9989826.
[10] Zachary's karate club: social network of friendships between 34 members of a karate club at a US university in the 1970s. Please cite W. W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33, 452-473 (1977).
[11] American College football: network of American football games between Division IA colleges during regular season Fall 2000. Please cite M. Girvan and M. E. J. Newman, Proc. Natl. Acad. Sci. USA 99, 7821-7826 (2002).
[12] Astrophysics collaborations: weighted network of co-authorships between scientists posting preprints on the Astrophysics E-Print Archive between Jan 1, 1995

and December 31, 1999. Please cite M. E. J. Newman, Proc. Natl. Acad. Sci. USA 98, 404-409 (2001).

[13] A. Katal, M. Wazid and R. H. Goudar, "Big data: Issues, challenges, tools and Good practices," 2013 Sixth International Conference on Contemporary Computing (IC3), Noida, 2013, pp. 404-409.

doi: 10.1109/IC3.2013.6612229

[14] JGraphT. https://jgrapht.org/ (last visited on July 2019)

[15] Weka. https://www.cs.waikato.ac.nz/ml/weka/ (last visited on July 2019)

[16] Yixin Chen , Li Tu, Density-based clustering for real-time stream data, Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, August 12-15, 2007, San Jose, California, USA [doi>10.1145/1281192.1281210]

[17] On Clustering Graph Streams, Charu C. Aggarwal, Yuchen Zhao, and Philip S. Yu Proceedings of the 2010 SIAM International Conference on Data Mining. 2010, 478-489

[18] JGraph : https://github.com/jgraph/

[19] Graph Strem : http://graphstream-project.org/

[20] Girvan M. and Newman M. E. J., Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99, 7821–7826 (2002)

[21] https://www.indeed.es/salaries/Investigador/a-Salaries