Escuela Técnica Superior de Ingeniería Informática

Grado en Ingeniería del Software

Curso 2022-2023

Detección de comunidades en redes sociales

Autor: Antonio Agudo Esperanza

Tutores: Jesús Sánchez-Oro Calvo

Abraham Duarte Muñoz

Agradecimientos

Quisiera aprovechar este espacio para expresar mi más sincero agradecimiento a todas las personas que contribuyeron de manera significativa a la realización de este Trabajo de Fin de Grado.

En primer lugar, quiero agradecer a mis tutores, por su guía experta y apoyo constante a lo largo de todo el proceso de investigación. Su sabiduría y conocimientos han sido fundamentales para el desarrollo de este trabajo, y su dedicación y orientación han sido invaluables.

También quiero mencionar a mis compañeros y amigos que me brindaron su apoyo durante todo el proceso. Sus palabras de aliento, discusiones y comentarios fueron vitales para superar los desafíos y mantenerme motivado.

No puedo olvidar agradecer a mi familia, quienes han sido mi mayor apoyo y motivación a lo largo de mi carrera académica. Su amor incondicional, paciencia y comprensión han sido fundamentales para llegar a este momento, sobre todo agradecer a mis hermanas por su constante motivación y exigencia hacia mi persona para alcanzar mis objetivos. Su entusiasmo y empuje me ha impulsado a superar mis límites y obtener resultados sobresalientes.

Finalmente me gustaría expresar mi reconocimiento a mi universidad y a todos los profesionales que contribuyeron a mi formación académica. Su compromiso con la excelencia educativa y su dedicación para brindar los recursos necesarios han sido determinantes en mi crecimiento personal y profesional.

Este trabajo es el resultado de un esfuerzo colectivo y no habría sido posible sin la contribución de todas estas personas. Mi más sincero agradecimiento a cada uno de ustedes por haberme acompañado en este viaje académico y por su valiosa ayuda en la realización de este Trabajo de Fin de Grado.

¡Gracias a todos!

Resumen

El problema de detección de comunidades se refiere a la identificación de agrupaciones o comunidades de nodos en una red o grafo. Una comunidad se define como un conjunto de nodos dentro del grafo que están más densamente conectados entre sí que comparado con el resto del grafo.

El objetivo principal de la detección de comunidades es dividir un grafo en subconjuntos de nodos que tengan conexiones más fuertes entre sí que con nodos externos.

Este problema es conocido como un problema NP-completo, lo que significa que no existe un algoritmo eficiente que pueda encontrar la solución óptima en todos los casos en un tiempo razonable. Por lo tanto, se utilizan diferentes enfoques y técnicas, tanto exactas como aproximadas para abordar este problema.

En este trabajo se ha empleado 2 algoritmos para la clasificación inicial de las comunidades.

Por otro lado, se ha empleado una red neuronal como modelo de aprendizaje automático para la detección de comunidades. Se estructura con capas que procesan las características de los nodos y aprenden representaciones y relaciones relevantes del grafo. Se han utilizado capas lineales y convolucionales para realizar las operaciones de procesamiento y propagación de la información en el grafo.

La combinación de una red neuronal y estos algoritmos de clasificación permite aprovechar las capacidades de aprendizaje y generalización de la red neuronal, mientras se utiliza el conocimiento y las estrategias de búsqueda de las metaheurísticas para mejorar la calidad de la detección de comunidades. Esto puede resultar en una solución más precisa y efectiva para el problema.

Esta combinación de enfoques permite aprovechar las fortalezas de ambos métodos y mejorar la capacidad de clasificación y detección de comunidades en el grafo.

Palabras clave:

- Detección de comunidades
- Red neuronal
- Algoritmo bio-inspirado
- Grafo
- NetworkX
- DGL (Deep Graph Library)
- Metaheurísticas (Louvain, ACO)
- Características de los nodos
- Optimización
- Análisis de redes
- Algoritmo de Louvain
- Ant Colony Optimization

Índice de contenidos

Índice de figuras

Índice de pseudocódigos

Índice de tablas

1.Introducción

En esta sección se va a introducir el problema de la detección de comunidades con el objetivo de que al lector le sea más fácil comprender el contexto del problema que se va a afrontar y en consecuencia la solución que se propone del mismo. También se va a proporcionar una revisión de la literatura y sitios consultados para la resolución de este problema además de un esquema general del documento.

1.1 Contexto y alcance

La detección de comunidades (Liu) en grafos es un problema fundamental en el análisis de grafos. Los grafos se encuentran en una amplia variedad de dominios, desde terrenos como las redes sociales, las redes biológicas, las redes de transporte y las redes de colaboración científica hasta terrenos más simples como puede ser una competición de karate, por ejemplo. Estas redes están compuestas por nodos y conexiones entre ellos, su estructura y características puede revelar información subyacente de gran importancia.

Tiene múltiples objetivos y aplicaciones. En primer lugar, permite revelar la estructura interna de la red identificando grupos de nodos que interactúan de manera intensa y formando subredes cohesivas. Esto proporciona información sobre la organización, jerarquía y modularidad de la red. Además, al asignar nodos a comunidades, se puede caracterizar la función y el comportamiento de los nodos, lo que facilita la inferencia de información sobre su rol y comportamiento en la red. Esta caracterización es útil en diversas áreas, como la segmentación de usuarios en redes sociales, la clasificación de genes en redes biológicas o la identificación de roles clave en redes de colaboración. La detección de comunidades también mejora la comprensión de la red al proporcionar perspectivas detalladas y comprensibles de su estructura y funcionamiento, lo que facilita el análisis, la interpretación de la información y la toma de decisiones informadas. Por último, tiene aplicaciones prácticas en la segmentación de mercados, la optimización de la distribución de recursos, la detección de anomalías y la

recomendación de contenido personalizado, al identificar grupos de nodos con características similares y permitir el diseño de estrategias específicas para cada comunidad.

Este Trabajo de Fin de Grado (TFG) se centrará en varias actividades principales. En primer lugar, se recopilarán conjuntos de datos de redes complejas reales o se generarán grafos sintéticos que representen diferentes estructuras de redes complejas. A continuación, se realizará el preprocesamiento de los datos, transformando los grafos en un formato adecuado para su análisis y extrayendo características relevantes de los nodos y las aristas.

Luego se procederá a la implementación de una arquitectura de red neuronal específicamente diseñada para aprender a detectar comunidades en los grafos. Se llevará a cabo el entrenamiento de la red neuronal utilizando los datos recopilados y preprocesados.

Además, se implementarán algoritmos bio-inspirados y algoritmos voraces como el "Ant Colony Optimization" también conocido por sus siglas ACO y Louvain respectivamente, con el objetivo de realizar una clasificación inicial de comunidades y mejorar los resultados obtenidos por la red neuronal.

Finalmente, se llevará a cabo un análisis e interpretación exhaustiva de los resultados. Se estudiarán las comunidades detectadas en las redes complejas y se analizará la influencia de diferentes técnicas y parámetros en los resultados obtenidos. Este análisis permitirá obtener conocimientos más profundos sobre las características de las comunidades en las redes complejas y su relevancia en el contexto del problema abordado.

En resumen, el proyecto se centrará en el desarrollo e implementación de técnicas de aprendizaje automático y algoritmos bio-inspirados para la detección de comunidades en redes complejas. Se explorarán diferentes enfoques y configuraciones para posteriormente evaluar los resultados obtenidos, con el objetivo de obtener una buena solución para cada caso en el problema.

Se quedará fuera del alcance de este proyecto la implementación de algoritmos avanzados desde cero ya que se utilizarán algoritmos ya existentes como puede ser Louvain y ACO. También se quedará fuera del alcance la optimización de hiperparámetros dentro de la red aunque se realicen ajustes en estos parámetros no se analizarán de manera exhaustiva como tampoco existirá una implementación de manera visual de las comunidades obtenidas.

1.2 Definición formal

1.2.1 Grafo dirigido

Un grafo dirigido $G = (V, E)$ consiste en un conjunto no vacío V de nodos y de un conjunto E de aristas dirigidas. Esta dirección es asociada con un par ordenado de nodos. Una arista dirigida es asociada con un par ordenado (u, v) donde el nodo u es el inicio y v es el final de la arista.

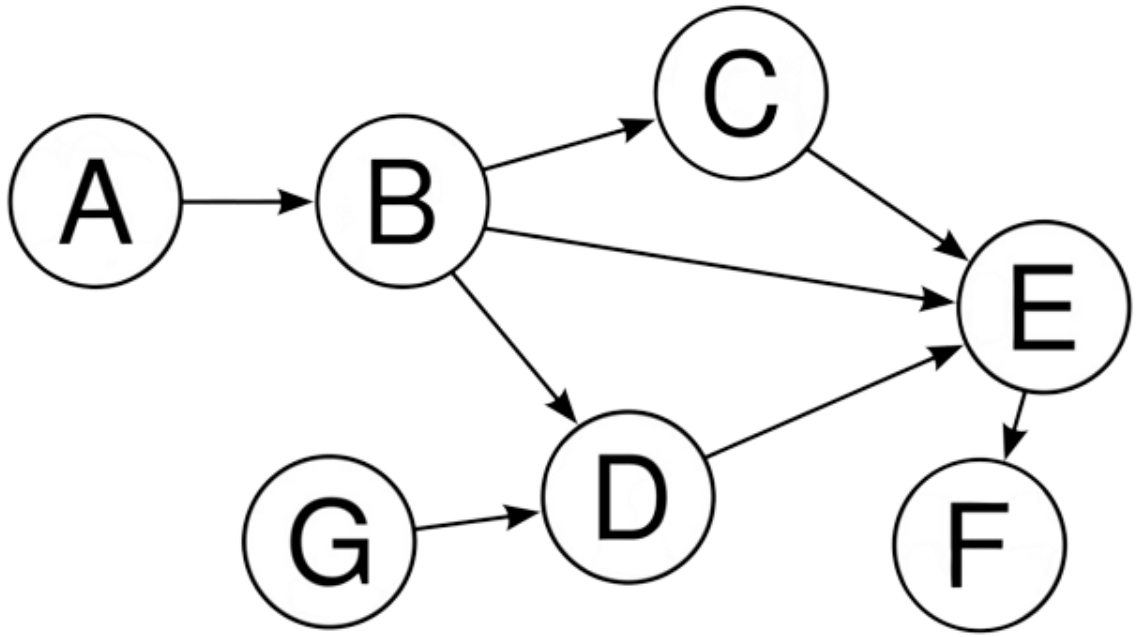


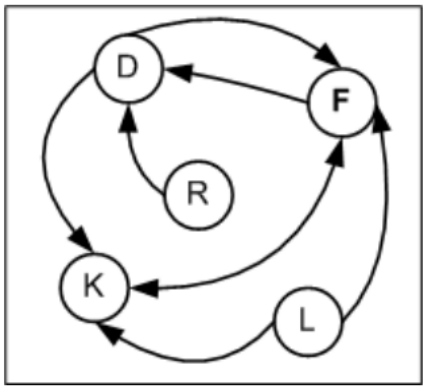
Figura 1: Ejemplo de Grafo dirigido

1.2.2 Adyacencia

Sea $G = (V, E)$ un grafo, donde V es el conjunto de vértices y E es el conjunto de aristas. La relación de adyacencia se define mediante una función o matriz de adyacencia A, que representa las conexiones o relaciones entre los vértices en el grafo. (Ant Colony Optimization, 2023)

Si tenemos n vértices en el grafo, la matriz de adyacencia A es una matriz cuadrada de tamaño n x n, donde cada entrada A[i, j] indica si hay una arista que conecta los vértices i y j.

En un grafo dirigido, donde las aristas tienen una dirección específica, la matriz de adyacencia puede ser asimétrica, es decir, A[i, j] puede ser diferente de A[j, i]. Esto indica que la adyacencia es unidireccional.



	D	F	K	L	R
D	0	1	1	0	0
F	1	0	1	0	0
K	0	1	0	0	0
L	0	1	1	0	0
R	1	0	0	0	0

Matriz de adyacencia

Figura 2: Matriz de adyacencia

1.2.3 Camino

Dado un grafo $G = (V, E)$, donde V es el conjunto de vértices y E es el conjunto de aristas, un camino en G es una secuencia de vértices y aristas alternados de la forma:

$$P = (v1, v2, v3, \dots, vk - 1, vk)$$

Donde v_1, v_2, …, v_k son vértices distintos del grafo G

La secuencia debe cumplir las siguientes condiciones:

1. Cada arista en la secuencia está conectada a los vértices adyacentes en la secuencia. Es decir, si $ei = (vi, vi + 1)$, entonces $vi\ y\ vi + 1$ son vértices adyacentes en el grafo G.

2. Cada vértice en la secuencia, excepto posiblemente el primero y el último, aparece exactamente dos veces en la secuencia, una vez como vértice de inicio de una arista y otra vez como un vértice final de una arista.

3. El primer vértice v1 en la secuencia es el vértice de inicio del camino, y el último vértice v_k es el vértice final del camino.

Un camino puede ser de longitud cero si consiste en un solo vértice sin aristas. Si existe un camino entre dos vértices u y v en un grafo, se dice que u y v son alcanzables o están conectados por ese camino.

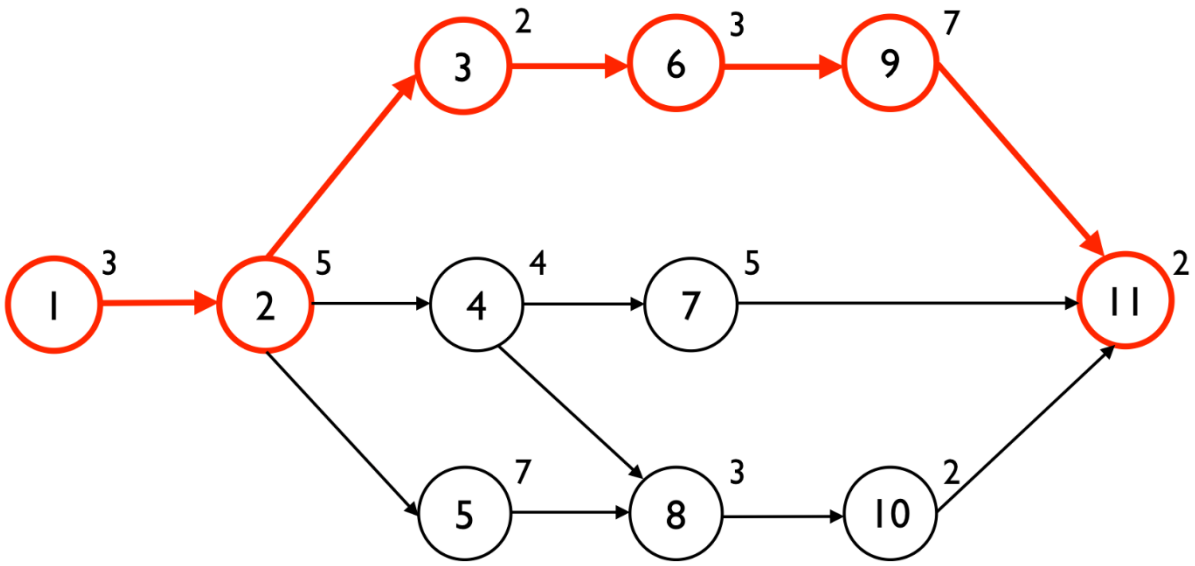


Figura 3: Ejemplo de camino desde el nodo 1 al nodo 11

1.2.4 Subgrafo

Sea G = (V, E) un grafo no dirigido. G_x = (X, E_x) es un subgrafo de G (inducido por X) si y sólo si $X \subseteq V \; y \; E_x \leq (X \; x \; X) \cap E$; es decir, si contiene un subconjunto de nodos de G y las aristas correspondientes.

Recíprocamente, el grafo G es un supergrafo de G_x si G_x es un subgrafo de G. Algunas veces, la condición de subgrafo hace referencia sólo a la condición de $E_x \subseteq E$ y un subgrafo inducido será aquel que cumpla la definición de subgrafo.

Un grafo G = (V, E) es un grafo completo si incluye todas las aristas posibles. Formalmente el grafo completo K_n de nodos n se define como:

$$K_n = V\ x\ V - \{(v, v) | v\ \in V\}$$

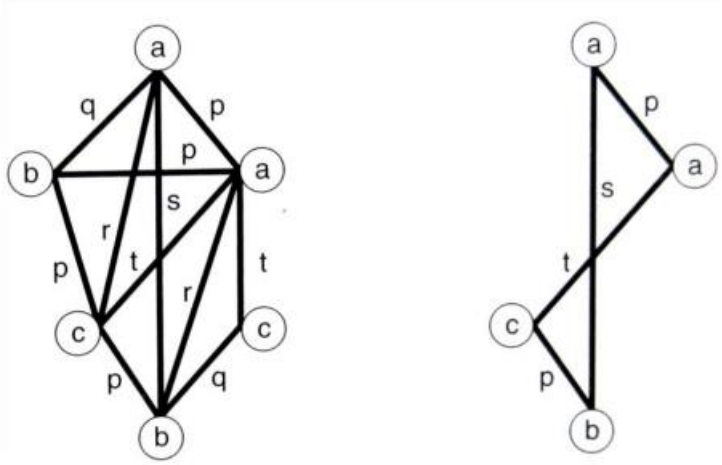


Figura 4: Ejemplo de subgrafo

1.2.5 Comunidad

La definición formal y matemática de comunidad en un grafo es un concepto que busca identificar subconjuntos de vértices en un grafo que están más densamente conectados entre sí que con los vértices fuera del subconjunto. Una comunidad en un grafo representa un grupo de nodos que exhiben una estructura de interconexión interna más fuerte que la conexión con los nodos fuera del grupo.

Formalmente, consideramos un grafo G = (V, E), donde V es el conjunto de vértices y E es el conjunto de aristas. Una comunidad en el grafo puede definirse de diferentes maneras, dependiendo de la métrica o algoritmo utilizado para su detección, el enfoque que se ha seguido en este trabajo ha sido basada en la modularidad cuya definición sería la siguiente:

$$Q = \left(\frac{1}{2m}\right) * \sum_{i=0}^{n}\sum_{j=0}^{n} e_{ij} - \frac{(d_i * d_{j)}}{2m} * \delta(C_i, C_j)$$

Donde:

- Q es la modularidad de la partición del grafo
- e_{ij} es el número de aristas entre los vértices i y j.
- d_i y d_j son los grados de los vértices i y j, respectivamente (es decir, el número de aristas que los conectan).
- m es el número total de aristas del grafo.
- $\delta(C_i, C_j)$ es un función delta que es 1 si C_i, C_j son iguales (es decir, los vértices i y j pertenecen a la misma comunidad) y 0 en caso contrario.

Esta fórmula de modularidad es comúnmente utilizada para evaluar la calidad de una partición en comunidades en un grafo. Cada término de la suma representa la contribución de una arista especifica a la modularidad total, teniendo en cuenta tanto las conexiones internas dentro de las comunidades como las conexiones esperadas al azar. Al maximizar la modularidad, se busca encontrar una partición que tenga una conexión interna más fuerte que las conexiones esperadas al azar.

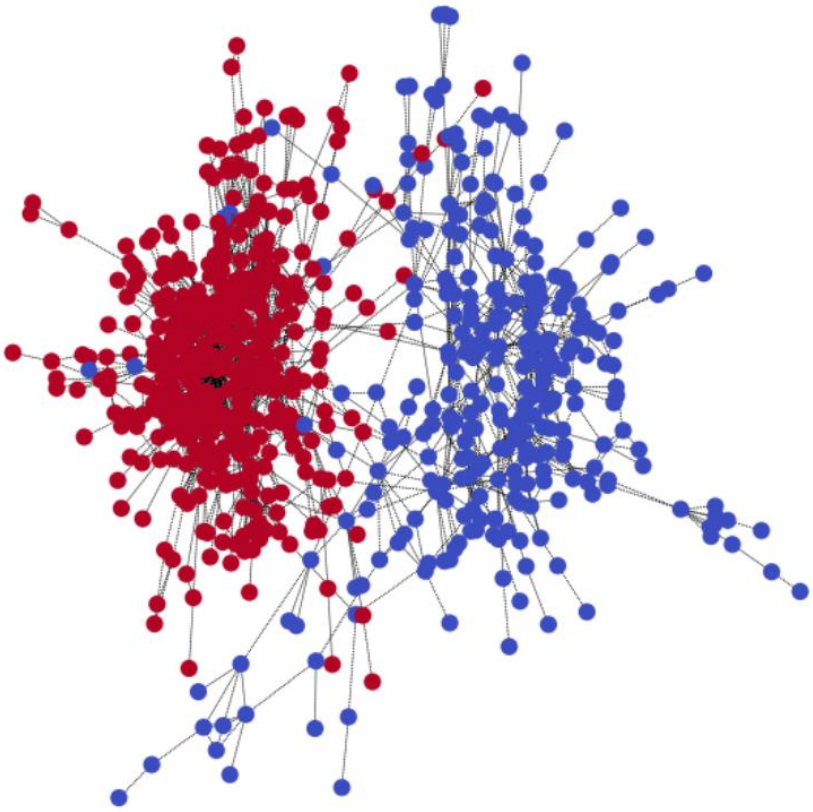


Figura 5: Ejemplo de grafo con 2 comunidades diferenciadas en color rojo y azul

1.2.6 Modularidad

La modularidad es una medida que evalúa la calidad de la partición de un grafo en comunidades. Matemáticamente, la modularidad Q se define como la diferencia entre la fracción de aristas que se encuentran dentro de las comunidades y la fracción esperada de aristas dentro de las comunidades elegidas aleatoriamente en la red.

Formalmente utilizamos la función de modularidad previamente mostrada.

La modularidad compara la cantidad de aristas dentro de las comunidades con la cantidad esperada si las aristas se distribuyeran aleatoriamente. Un valor de modularidad cercano a 1 indica que la partición del grafo en comunidades es significativa, mientras que un valor cercano a 0 indica que la partición no es significativa.

1.2.7 Red neuronal

Una red neuronal, se puede definir matemáticamente como una función compuesta que mapea una entrada *x* a una salida *y*. Está compuesta por múltiples capas de neuronas interconectadas, donde cada neurona realiza una operación de combinación lineal seguida de una función de activación no lineal.

Formalmente, para una red neuronal con L capas, la salida y se calcula como:

$$y = fL(fL-1(\dots(f2(f1(x \cdot W1 + b1) \cdot W2 + b2) \dots) \cdot WL + bL)$$

Donde:

- f_i representa la función de activación en la capa i
- W_i representa los pesos sinápticos de la capa i
- b_i representa los sesgos de la capa i

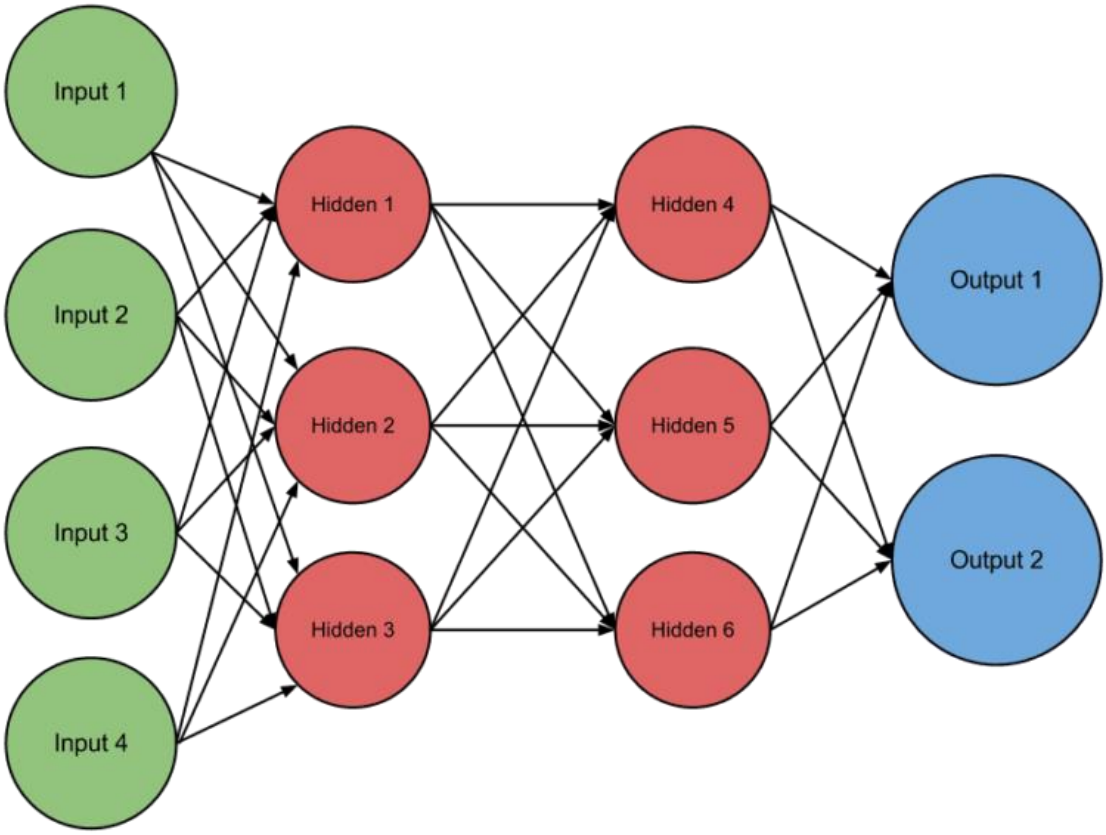


Figura 6: Ejemplo de red neuronal con 4 parámetros de entrada, 2 capas ocultas y 2 salidas

A su vez, cada neurona que forma parte de la red se puede definir como un componente que toma datos de entrada, realiza operaciones matemáticas en ellos utilizando pesos y una función de activación, y produce una salida.

1.3 Revisión del estado del arte

El problema de detección de comunidades en grafos ha sido ampliamente estudiado en la literatura científica, y existen numerosos enfoques y técnicas desarrolladas para abordar este desafío.

Algunos de los más utilizados pueden ser los métodos basados en optimización como podrían ser el algoritmo de Louvain o el algoritmo de Newman-Girvan (Newman, Networks: An Introduction) que buscan maximizar la modularidad del grafo o minimizar una función de coste asociada a la partición del grafo. Son muy eficientes y capaces de detectar comunidades en grafos grandes, existen metaheurísticas cuyo enfoque está basado en el "Variable Neighborhood Search" (VNS) (Sergio Pérez-Peló) que aprovecha la combinación de calidad y diversidad de un procedimiento constructivo inspirado en el "Greedy Randomized Adaptative Search Procedure" (GRASP)

También se han propuesto otros métodos como los detección espectral (Luxburg) que aprovechan las propiedades espectrales de la matriz de adyacencia, métodos de agrupamiento que usan algoritmos como el de propagación de afinidad, métodos probabilísticos (Friedman) que usan modelos de inferencia estadística para asignar los nodos a las comunidades en función de la probabilidad de conexiones entre ellos, también se han usado métodos de aprendizaje automático que usan enfoques como el de aprendizaje no supervisado y el aprendizaje por refuerzo.

En la actualidad se están explorando nuevos enfoques como podría ser el enfoque mixto donde se está haciendo uso de técnicas de Deep Learning y modelos probabilísticos para mejorar la precisión y la escalabilidad de los algoritmos de detección de comunidades. Además, se están investigando aspectos más desafiantes como la detección de comunidades en grafos dinámicos y la integración de atributos adicionales de los nodos para mejorar la calidad de las particiones, además se están siguiendo enfoques multiobjetivo que se basan en funciones de optimización como por ejemplo la función de "Negative Ratio Association" (NRA) y la función de "Ratio Cut" (RC) cuyos objetivos se han probado estar en conflicto.

1.4 Estructura del documento

Este documento se estructura de la siguiente manera:

- En el Capítulo 2 se describirá cuáles han sido los objetivos de este trabajo, por un lado, se explicará el objetivo principal del mismo y que se ha intentado conseguir con él y por el otro los objetivos secundarios alcanzados durante la realización de este.

- En el Capítulo 3 se presentan las diferentes propuestas algorítmicas donde se explicará detalladamente los algoritmos utilizados y la estructura de la red neuronal utilizada.

- En el Capítulo 4 se analiza el desarrollo informático del trabajo explicando la estructura del proyecto y los diferentes componentes que se han desarrollado en el mismo.

- En el Capítulo 5 se mostrarán los distintos experimentos con las distintas configuraciones de parámetros y algoritmos utilizados. Además, también se mostrará el resultado comparativo de las configuraciones para así comprobar cuál es la configuración óptima obtenida del problema.

- Por último, en el Capítulo 6 se mostrarán las conclusiones derivadas del trabajo y posibles ideas de mejora para el mismo.

2.Objetivos

En este capítulo se describen los objetivos que se han tratado conseguir en este trabajo, se ha dividido entre el objetivo principal, que es la razón principal del desarrollo de este trabajo y los objetivos secundarios, que son las metas que se han intentado conseguir derivadas del objetivo principal.

2.1 Objetivo principal

El objetivo principal de este trabajo es el diseño e implementación de diferentes configuraciones para solucionar el problema de detección de comunidades con un enfoque mixto basado en métodos basados en la optimización como Louvain o el algoritmo de las hormigas y métodos de Deep Learning.

2.2 Objetivos secundarios

Como objetivos secundarios de este trabajo se han elegido los siguientes:

- Adquirir conocimientos sobre el desarrollo de redes neuronales y Deep Learning.
- Adquirir conocimientos sobre algoritmos complejos.
- Mejorar las habilidades de programación adquiridas en otras asignaturas cursadas
- Profundizar el conocimiento en el lenguaje de programación Python.

- Conocer el correcto uso y funcionamiento de librerías de Python como DGL, Pytorch, NetworkX, Pandas y MatplotLib.

- Mejorar las habilidades en el trabajo de grafos puesto que se trata de una estructura compleja y muy demandada.

- Conocer métodos para ajusta parámetros en redes neuronales y algoritmos.

3.Descripción algorítmica

Este capítulo presenta las propuestas algorítmicas planteadas para solucionar el problema. La descripción algorítmica del proyecto se puede dividir en 4 etapas principales: preprocesamiento de datos, detección inicial de comunidades basada en algoritmos de optimización, detección de comunidades a través de redes neuronales y análisis e interpretación de resultados.

3.1 Preprocesamiento de datos

3.1.1 Carga de datos

Se cargan conjuntos de datos de redes complejas o se generan grafos que representen diferentes estructuras de redes complejas. Para este trabajo, se eligieron 3 tipos de grafo.

El grafo de Barbell (Barbell Graph, s.f.) es un tipo de grafo que consta de dos comunidades unidas por un camino, tiene este nombre “barbell” (mancuerna en español) debido a su forma semejante a una barra con 2 pesas en los extremos. Formalmente se define de la siguiente manera:

Sea K_n una comunidad completa de tamaño n y P_m el camino de longitud m. Un grafo de Barbell denotado como $B_{n,m}$ consiste en la unión de 2 comunidades K_n en los extremos y unidos por un camino P_m en el medio.

Graficamente se representa de la siguiente manera:

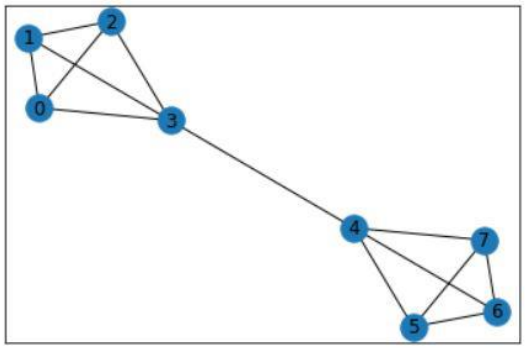


Figura 7: Ejemplo de Grafo Barbell

El siguiente grafo con el que se ha trabajado es el grafo LFR-Benchmark (LFR_benchmark_graph, s.f.) creado por Lancichinetti-Fortunato-Radicchi (de ahí su nombre LFR), es un tipo de grafo sintético utilizado en la evaluación y comparación de algoritmos de detección de comunidades.

Construye mediante asignación de comunidades a nodos y la generación de conexiones entre ellos siguiendo ciertas reglas. Algunas de sus características claves son:

1. Distribución de grado: Los nodos en el grafo siguen una distribución de grado de potencia, lo que significa que hay pocos nodos con grados altos y muchos nodos con grados bajos.
2. Tamaño y número de comunidades: El grafo tiene un número predefinido de comunidades, y cada comunidad tiene un tamaño específico que puede variar. Esto permite controlar la estructura de las comunidades y su distribución en el grafo.
3. Solapamiento de comunidades: El LFR-Benchmark también permite la generación de comunidades superpuestas, donde un nodo puede pertenecer a más de una comunidad.
4. Asignación de conexiones intercomunitarias: Se establecen conexiones entre nodos de diferentes comunidades para simular la estructura de red del mundo real, donde existen interacciones entre grupos distintos.

Este grafo ha sido clave durante el desarrollo del proyecto ya que ha permitido comparar los algoritmos de detección de comunidades en un entorno realista.

Su representación gráfica sería la siguiente:

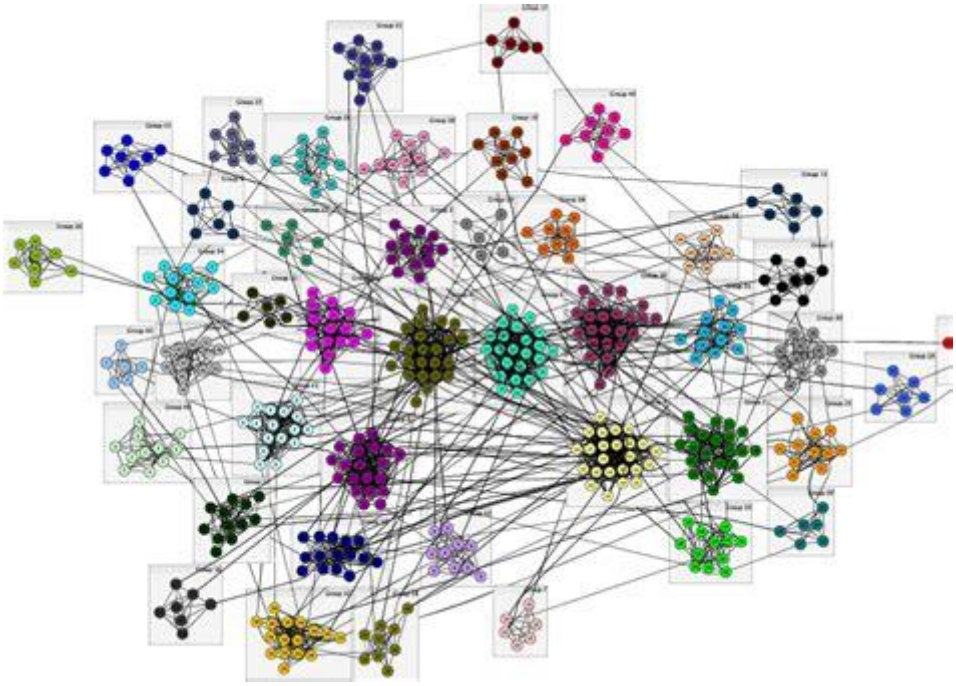


Figura 8: Ejemplo de Grafo LFR_Benchmark

Por último, se ha permitido la creación de grafos aleatorios generados por el usuario, para ello se toman las aristas y “features” proporcionadas por el usuario y se va creando su propio grafo personalizado dando pie al siguiente resultado:

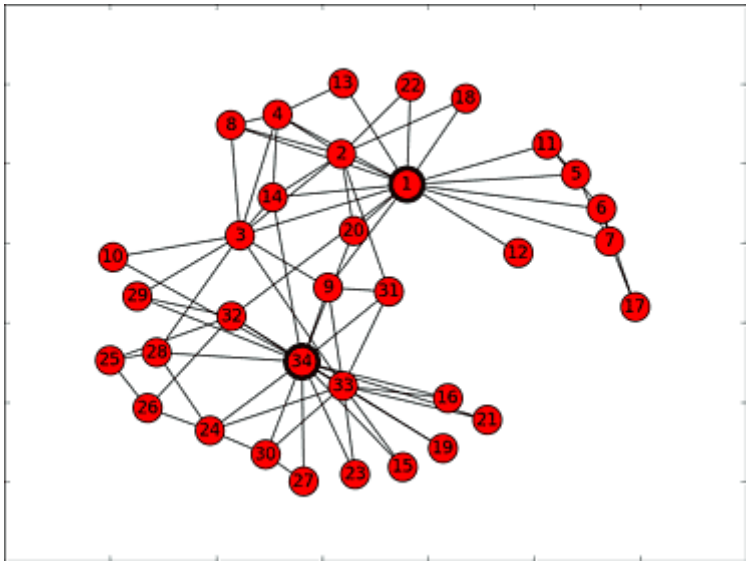


Figura 9: Ejemplo de grafo creado por el usuario

3.1.2 Extracción de características

Se extraen las características más relevantes de los nodos y las aristas que servirán como entrada para la red neuronal.

Es muy importante que estas características sean propiedades intrínsecas de la red y que no se deban a su fuente ya que el objetivo es trabajar con todo tipo de redes.

Para ello se hizo una fase inicial que obtenía números aleatorios como característica del grafo.

Luego se procedió a implementar a cada nodo el algoritmo de Dijkstra para saber la distancia de cada nodo a todos los nodos ya que está información podría ayudar a la red a detectar las comunidades puesto que en una comunidad la distancia entre los nodos será menor que con los nodos fuera de su comunidad.

Algorithm 1 Algoritmo de Dijkstra

Require: Grafo G, nodo inicial s
Ensure: Distancias mínimas $d[]$ desde s a todos los nodos

```
Inicializar d[] con infinito, excepto d[s] = 0
Inicializar cola de prioridad Q con pares (nodo, distancia)
Insertar (s, 0) en Q
while Q no está vacío do
    Extraer nodo u de Q con la distancia mínima
    for Cada vecino v de u do
        Calcular la nueva distancia nd = d[u] + peso(u, v)
        if nd < d[v] then
            Actualizar d[v] = nd
            Actualizar la prioridad de v en Q a nd
        end if
    end for
end while
Return d[]
```

Pseudocódigo 1: Algoritmo de Dijkstra

Posteriormente, se procedió a sacar la centralidad de cada nodo ya que este valor nos puede ser útil para saber la importancia de un nodo en la red. En concreto se usó la centralidad de intermediación que mide la frecuencia con la que un nodo actúa como puente en los caminos más cortos de los nodos.

Esta centralidad mide la importancia de un nodo v en términos de cuántos caminos más cortos pasan a través de ese nodo en relación con todos los caminos más cortos posibles entre los demás nodos.

Se puede definir matemáticamente de la siguiente manera:

Sea G = (V,E) un grafo no dirigido y conectado, donde V es el conjunto de nodos y E es el conjunto de aristas. La centralidad de intermediación B(v) de un nodo se calcula cómo:

$$B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Donde:

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

Algorithm 3 Algoritmo de Centralidad del nodo

Require: Grafo G, nodo objetivo
Ensure: Centralidad del nodo v

```
Inicializar centralidad[v] = 0
Inicializar pila S y conjunto visitados  Apilar v en S
Marcar v como visitado
while S no está vacío do
    Desapilar nodo u de S
    Incrementar centralidad[u] por 1
    for Cada vecino w de u do
        if w no ha sido visitado then
            Apilar w en S
            Marcar w como visitado
    return centralidad[v]  =0
```

Pseudocódigo 2: Algoritmo de Centralidad de nodo

3.2 Detección de comunidades con algoritmos de optimización

Se usaron diferentes algoritmos para hacer una clasificación inicial de los nodos de un subgrafo puesto que la intención es usar posteriormente una red neuronal supervisada por lo que tenemos que pasarle de antemano a qué comunidad pertenece cada nodo. Para ello se usaron 2 algoritmos diferentes Louvain y el algoritmo de las hormigas ya que tienen diferentes enfoques de resolver el problema.

3.2.1 Algoritmo de Louvain

El algoritmo de Louvain es uno de los algoritmos conocidos como voraces ya que busca encontrar una partición óptima de una red en comunidades a través de la maximización de la modularidad, que es una medida de la calidad de la partición de la red en comunidades.

En cada iteración, el algoritmo reasigno los nodos a comunidades de manera iterativa para mejorar la modularidad. El proceso continúa hasta que no se puede mejorar más la modularidad.

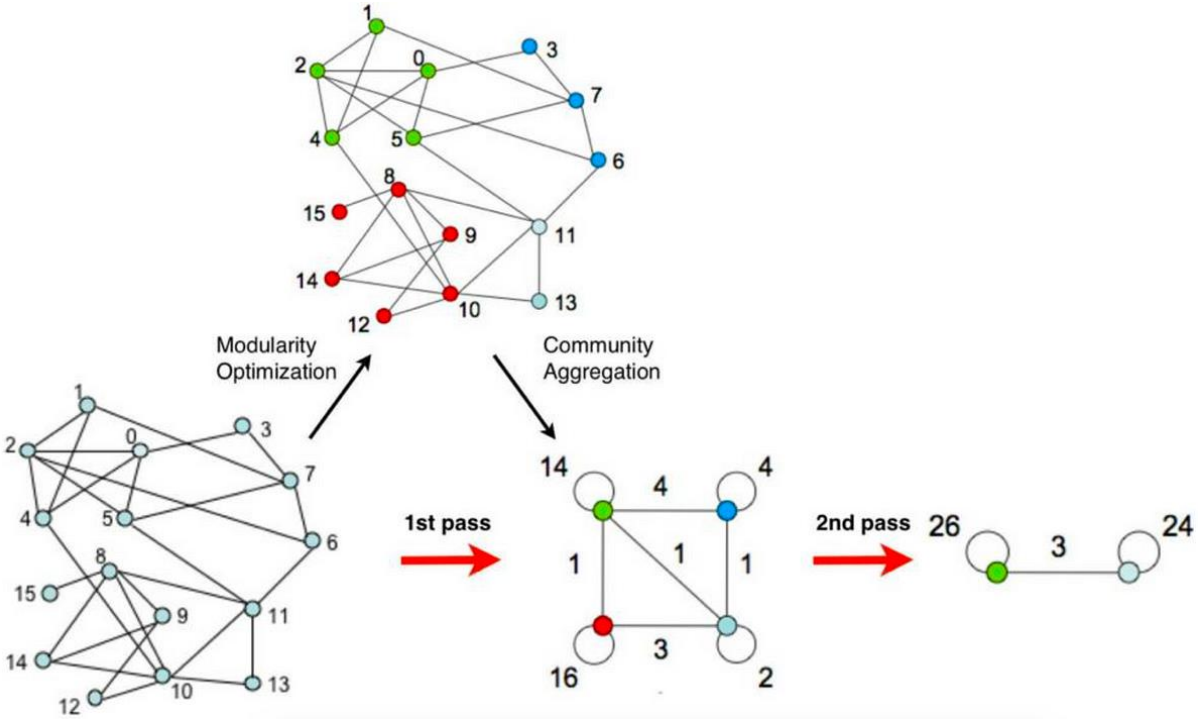


Figura 10: Ejemplo visual del algoritmo de Louvain

Este consta de 2 partes.

Primero se asigna a cada nodo de la red una comunidad, luego para cada nodo se calcula la diferencia de modularidad al removerlo de su comunidad y agregarlo a cada una de las comunidades de sus vecinos. Este valor se calcula 2 pasos: se saca el nodo de su comunidad original y es insertado en la comunidad de su nodo vecino, la ecuación resultante sería la siguiente:

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

En la segunda fase del algoritmo, se agrupa todos los nodos de una comunidad y se crea una nueva red donde los nodos son las comunidades de la fase anterior.

Algorithm 2 Algoritmo de Louvain

Require: Grafo G, número de iteraciones T
Ensure: Comunidades optimizadas

```
Inicializar cada nodo como una comunidad separada
for t = 1 hasta T do
    Calcular la ganancia de modularidad para cada nodo al moverlo a otras comunidades
    Mover cada nodo al vecino que maximice la ganancia de modularidad
end for
Realizar la fusión de comunidades con el mismo identificador
return Comunidades optimizadas
```

Pseudocódigo 3: Algoritmo de Louvain

3.2.2 Algoritmo de las hormigas

El algoritmo de la colonia de hormigas (Ant Colony Optimization, ACO) [2] para la detección de comunidades en redes complejas se basa en el comportamiento de las hormigas reales que buscan alimentos y dejan rastros químicos para comunicarse entre sí

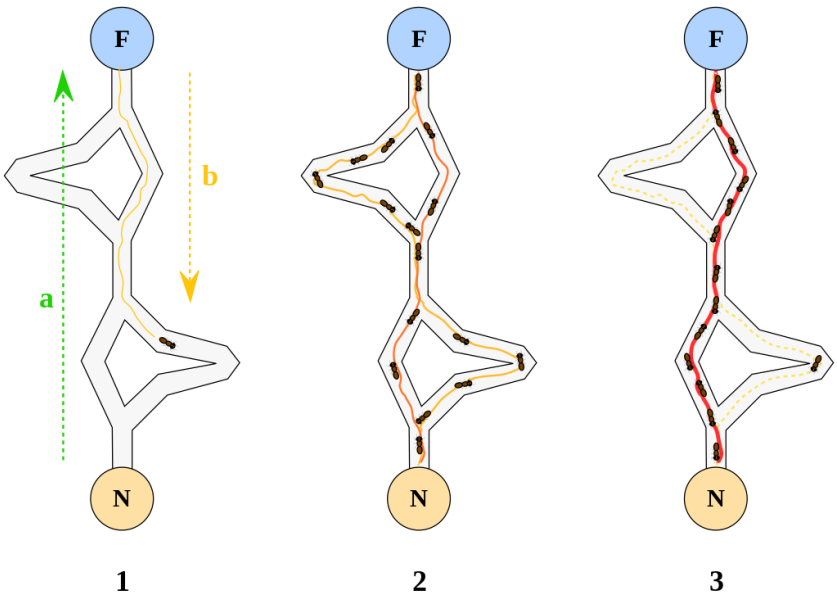


Figura 11: Funcionamiento de las hormigas en la vida real

Para la detección de comunidades el algoritmo primero inicializa los parámetros como el número de hormigas, el número máximo de iteraciones y la tasa de evaporación de feromonas para posteriormente inicializar el valor de feromonas en todas las aristas del grafo.

Posteriormente se empiezan a generar las soluciones, por cada hormiga de la colonia se elige un nodo inicial de manera aleatoria y construye una solución recorriendo el grafo seleccionando las aristas basadas en la probabilidad entre las feromonas y las heurísticas locales siguiendo la siguiente fórmula:

$$p_{xy}^{k} = \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum \tau_{xy}^{\alpha}\, \tau_{xy}^{\beta}}$$

Donde:

- k es la hormiga
- x es el estado actual
- y es el estado al que se va a mover
- τ_{xy} es la cantidad de feromonas depositadas en la transición del estado x a y
- η_{xy} es el "atractivo" del movimiento computado por alguna heurística, en nuestro caso se usó la modularidad
- α es un parámetro de control de la influencia de τ_{xy}
- β es un parámetro de control de la influencia de η_{xy}

A continuación, se actualizan los valores de las feromonas en todas las aristas según las soluciones construidas por las hormigas para ello se actualiza la regla de evaporación, dictada por la fórmula: $\tau(e) = (1 - \rho) * \tau(e)$ donde τ(e) es el valor de feromonas en la arista e y ρ es la tasa de evaporación de feromonas. También se actualiza la regla del depósito de feromonas dictada por la fórmula: $\Delta\tau(e) = Q/L$ donde $\Delta\tau(e)$ es la cantidad de feromonas depositadas en la arista e, Q es una constante que representa la cantidad de feromonas depositadas por una hormiga y L es la calidad de la solución construida por la hormiga.

Finalmente se evalúa la calidad de la solución con una función de evaluación como en nuestro caso la modularidad y se guarda la mejor solución hasta el momento. Si se cumple el criterio de terminación ya sea porque se ha alcanzado el número máximo de iteraciones o porque no se

produce una mejora en la calidad de las soluciones, se finaliza el algoritmo y se devuelve la mejor solución encontrada.

Algorithm 3 Algoritmo de Hormigas

Require: Grafo G, número de hormigas N, número de iteraciones T
Ensure: Mejor solución encontrada

```
Inicializar feromonas en todas las aristas del grafo
Inicializar las hormigas en nodos aleatorios
for t = 1 hasta T do
    for cada hormiga i do
        Seleccionar siguiente movimiento basado en las feromonas y la heurística
        Actualizar feromonas en la arista utilizada por la hormiga
    end for
    Actualizar la mejor solución encontrada
    Evaporar feromonas en todas las aristas
end for
return Mejor solución encontrada
```

Pseudocódigo 4: Algoritmo de Hormigas (ACO)

3.3 Detección de comunidades con redes neuronales

En el ámbito de la detección de comunidades en redes complejas, el uso de redes neuronales se ha convertido en una técnica prometedora.

Una red neuronal se define formalmente como un modelo matemático compuesto por unidades de procesamiento interconectada, que realizan operaciones de procesamiento y aprendizaje a través de conexiones ponderadas. Es capaz de capturar relaciones no lineales y realizar tareas complejas de procesamiento de información.

Se ha divido la implementación de la red en varios puntos, primero, se decidió la arquitectura de la red ya que es crucial el número de entradas, la cantidad de capas internas que tiene y la salida que se espera de esa red. Posteriormente se decidió el entrenamiento de la red neuronal donde se seleccionaba el conjunto de datos para entrenamiento, validación y evaluación. Por último, se hizo una configuración de los hiperparámetros para conseguir una mejor solución.

3.3.1 Arquitectura de la red

La arquitectura de la red es el diseño de las capas de neuronas y conexiones entre ellas que permite el procesamiento de la información. La red contará siempre con una capa de entrada que en nuestro caso son las características de cada nodo y una capa de salida que será el número de clases a las que puede pertenecer un nodo en concreto, entre estos resultados se elegirá el que tenga una probabilidad más alta, en nuestro caso se hizo una mejora para que contará de una capa intermedia cumpliendo así la definición de Deep Learning.

Primero se empezó con una red neuronal que simplemente contaba con la capa de entrada y la capa de salida. Este enfoque no daba ningún resultado y la red era incapaz de aprender puesto que no conseguía ponderar correctamente los pesos para un resultado óptimo.

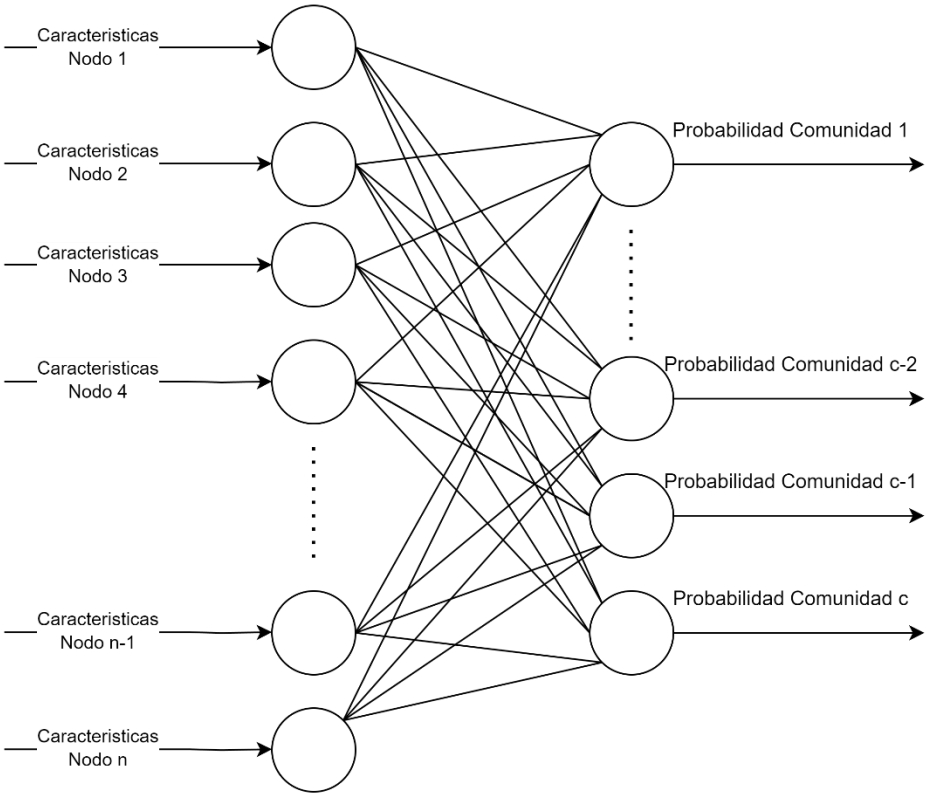


Figura 12: Primer modelo de red neuronal

Seguidamente se incluyó una capa de procesamiento cuyo objetivo era extraer la información más relevante de cada nodo en estas características

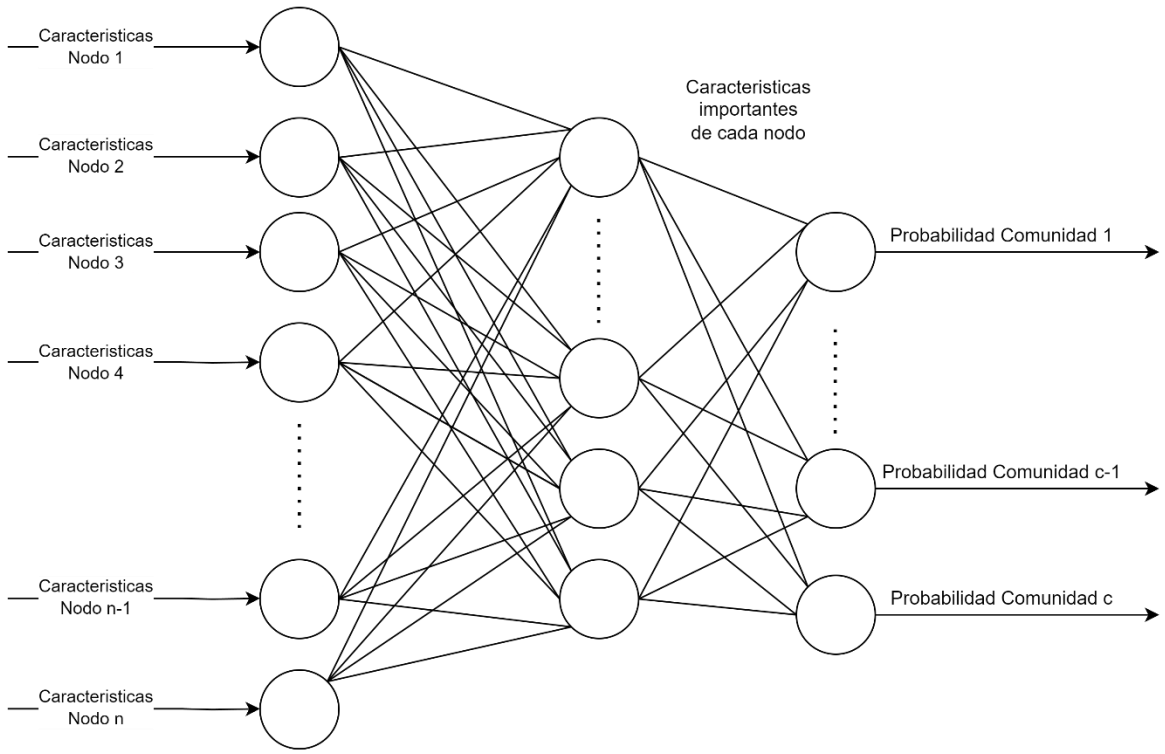


Figura 13: Evolución del modelo

Posteriormente se agregó una capa con el objetivo de que los nodos intercambiasen información entre ellos con el objetivo de conseguir que la red tenga más en cuenta la topología de la misma red. En una primera instancia, se planteó esta red como capa de entrada, pero al fijarse que perdía información de las características de los nodos se decidió dejar como una capa intermedia.

En la red se han aplicado capas convolucionales ya que el objetivo es que cada capa que representa la información de un nodo tenga la información de sus vecinos para conseguir captar similitudes.

Se usó de función de activación la función ReLU (Rectified Linear Unit), es una función no lineal que asigna valores negativos a cero y mantiene los valores positivos sin cambios. De función de perdida se introdujo la función de perdida de entropía cruzada (Cross-Entropy Loss), es una función muy utilizada en problemas de clasificación y por ese motivo se eligió para este trabajo.

3.4 Análisis e interpretación de resultados

Para la validación del resultado se ha usado la métrica conocido como NMI (Normalized Mutual Information) es una métrica comúnmente utilizada para evaluar la calidad de las agrupaciones o comunidades en problemas de detección de comunidades. El NMI mide la similitud entre 2 particiones una que se da de referencia y la que ha analizado la red.

El NMI calcula la cantidad de información mutua normalizada entre las dos particiones, teniendo en cuenta tanto la similitud entre los grupos asignados como la similitud esperada por azar. Proporciona un valor entre 0 y 1 donde 0 indica una falta de similitud entre las particiones y 1 indica una similitud perfecta.

La fórmula para calcular el NMI es la siguiente:

$$NMI = \frac{(2 * MI)}{\big(H(A) + H(B)\big)}$$

Donde:

- MI (Mutual Information) es la información mutua entre las particiones A y B
- H(A) y H(B) son las entropías de las particiones A y B, respectivamente

El NMI se utiliza para comparar diferentes algoritmos de detección de comunidades y evaluar su capacidad para encontrar agrupaciones similares a la partición de referencia. Un valor más alto de NMI indica una mejor calidad de las comunidades detectadas.

En este trabajo se ha utilizado de referencia para el NMI las comunidades asignadas por los algoritmos que se han comparado con las comunidades extraídas de las etiquetas de validación de la red.

3.5 Estructura final del proyecto

En el siguiente diagrama de flujo se puede apreciar el flujo de ejecución de la aplicación en el cuál, pasa por todos los componentes previamente nombrados.

Como se puede apreciar, se utilizan todos los componentes anteriormente descritos, primero hace una fase de selección del grafo a estudiar para posteriormente elegir el algoritmo de clasificación con el objetivo de darle un conjunto de muestra a la rede que puede usar tanto los nodos del grafo como entrada o sus características para finalmente analizar los resultados con la función NMI con las comunidades ya clasificadas.

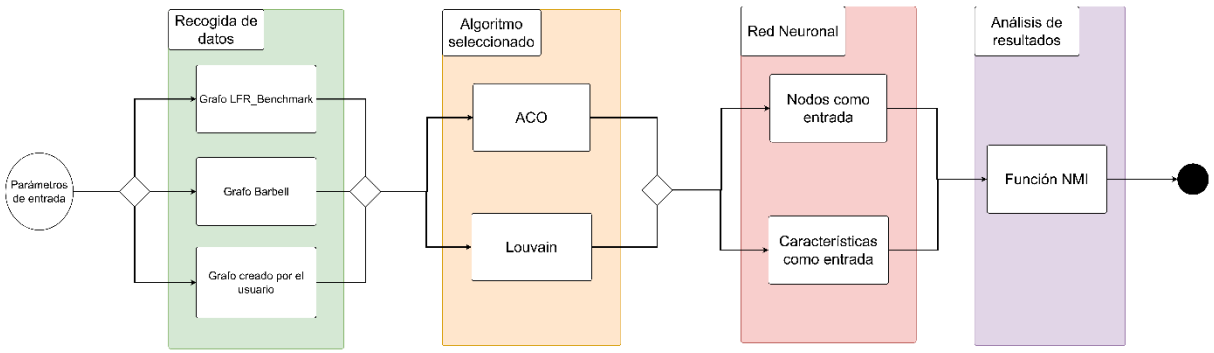


Figura 14: Diagrama de flujo de la aplicación

4.Descripción Informática

En este capítulo se describe la manera en la que el trabajo ha sido desarrollado, se explicará el flujo entre los diferentes ficheros a nivel de programación y se analizará el código y librerías usadas en el desarrollo de los componentes del Capítulo 3

El proyecto ha sido desarrollado en Python en su versión 3.10 y se ha aprovechado la versatilidad del propio lenguaje para tener un enfoque multiparadigma. Se ha utilizado el paradigma de orientación a objetos para desarrollar la red neuronal dado que la librería usada para implementar la red (Deep Graph Library) estaba diseñada para ser utilizada con este enfoque, mientra0s que se ha utilizado un enfoque más funcional para el desarrollo de los demás componentes del trabajo.

4.1 Metodología y Herramientas

Para la realización de este Trabajo de Fin de Grado se ha seguido una metodología Ágil basada en iteraciones con revisiones constantes para revisar los avances conseguidos.

Se ha seguido un enfoque incremental donde primero se realizó los componentes de la red neuronal y la generación de grafos de manera muy básica simplemente incorporando el grafo de Benchmark al ser el más sencillo, una red de una sola capa y una clasificación manual de la red sin uso de ningún algoritmo aprovechando la simpleza del grafo.

Luego se amplió para ir incluyendo más componentes como una métrica manual donde se compara las comunidades detectadas por la red y las detectadas manualmente obteniendo de esta manera el porcentaje de acierto, más tipos de grafo como el LFR_Benchmark que es un grafo más complejo, mejoras en la red para incluir más capas para un mejor procesamiento de la información y se testeó el cambio de estrategia cambiando la entrada de la red del número de nodos a un nuevo enfoque basado en la utilización de la lista de propiedades de cada nodo

obteniendo una mejora considerable en la red, además, se incorporó el análisis de los resultados usando la función NMI en vez de la comparación entre las etiquetas.

Por último, se incluyó un generador de grafos que pueden ser generados por el usuario usando un Excel, los algoritmos de Lovain y hormigas para la clasificación y se incluyó un módulo para exportar gráficas con los resultados obtenidos.

Por otra parte, se usó GitHub como gestor de versiones del proyecto dada su facilidad de manejo y por la familiaridad con el gestor, además, de esta manera cualquier otro usuario podrá acceder al código de este Trabajo de Fin de Grado si se desea inspeccionar o decide instalar el proyecto. Se empezó usando PyCharm para manejar el código, pero posteriormente se pasó a VisualStudioCode debido a problemas con el IDE al tratar de manejar el mismo las librerías utilizadas dando problemas causando que se tenga que reinstalar el proyecto.

4.2 Diseño e implementación

En esta sección se describen las decisiones de diseño tomadas durante el proyecto, se muestra la estructuración en archivos de este, además de la relación que existen entre ellos y la utilidad de cada uno de los archivos.

Se ha divido la estructura en módulos para mejorar la claridad del código, los 4 módulos principales son los que se han mostrado en el capítulo anterior.

El primer módulo sería Graph_construction su utilidad sería la construcción del grafo, este módulo se divide a su vez en tres archivos para cada tipo de grafo explicado anteriormente, para el grafo de Barbell y el grafo de Benchmark su implementación fue muy sencilla gracias a que se usó la librería NetworkX que usando una serie de parámetros es capaz de construir por si mismo la implementación del mismo, por otra parte, se usó la librería de Pandas para leer el csv donde se implementa el grafo que puede ser generado por parte del usuario; se encarga de leer 2 columnas del mismo (src,dst) y guarda el resto de columnas en una lista llamada “features”, después de este proceso, NetworkX se encarga de usar las 2 primeras columnas para establecer las aristas entre los vértices y se reserva la lista de features para ser incluida posteriormente como parte de la lista de características de la red neuronal.

El segundo módulo con el que se va a trabajar es Algorithms, dónde se ha incluido los algoritmos utilizados para este trabajo, se divide en 2 archivos, el primero, Ant_colony.py contiene el algoritmo de las hormigas explicado anteriormente, y el segundo, Louvain.py contiene 2 funciones principales, la primera función es la implementación manual del algoritmo de Louvain que por problemas de rendimiento, se sustituyó por la segunda función que es el algoritmo de Louvain implementado por NetworkX que aprovecha la topología del grafo para opitmizar el algoritmo, también se han incluido varios algoritmos que se usarán para obtener más información sobre la red; el primer algoritmo con este propósito es el algoritmo de Dijkstra que, al igual que con Louvain se usaron 2 implementaciones, una manual y otra implementada por la librería de NetworkX por temas de optimización; el segundo algoritmo implementado para este propósito es betweeness_centrality que calcula la centralidad de cada nodo, en esta función también se usó la implementación optimizada de NetworkX.

El tercer módulo es Neuronal_Network, que contiene la implementación de la red neuronal con la que se va a trabajar, para ello se ha usado la librería de Deep Graph Library ya que contiene una solución optimizada para el trabajo con grafos en redes de este estilo. Esta librería posee la capacidad de trabajar con 2 backends para el procesamiento de la información de la red, el primero sería haciendo uso de la librería PyTorch con la que se empezó a trabajar, aunque posteriormente se pasara a utilizar la otra librería de fondo TensorFlow debido a problemas en el desarrollo que se explicarán más adelante. Contiene un solo archivo llamado Network.py donde se han implementado 2 clases con 2 modelos de red diferentes, la primera contiene un primer modelo de red donde se usa de entrada cada nodo.

El último módulo utilizado es Analytics, el cuál obtiene las métricas del rendimiento del modelo utilizando la función normalized_mutual_information de la librería sklearn la cuál es la implementación de la función NMI que se ha usado para conseguir el rendimiento de la red, también se implementa la función show_results, la cual se encarga de mostrar visualmente el grafo y se implementa también una función para ver las gráficas de la función del coeficiente de perdida de la red y la cantidad de aciertos por época que ha tenido la misma. Para este trabajo, se usó la librería MatplotLib ya que permite de una manera sencilla construir gráficas de resultados.

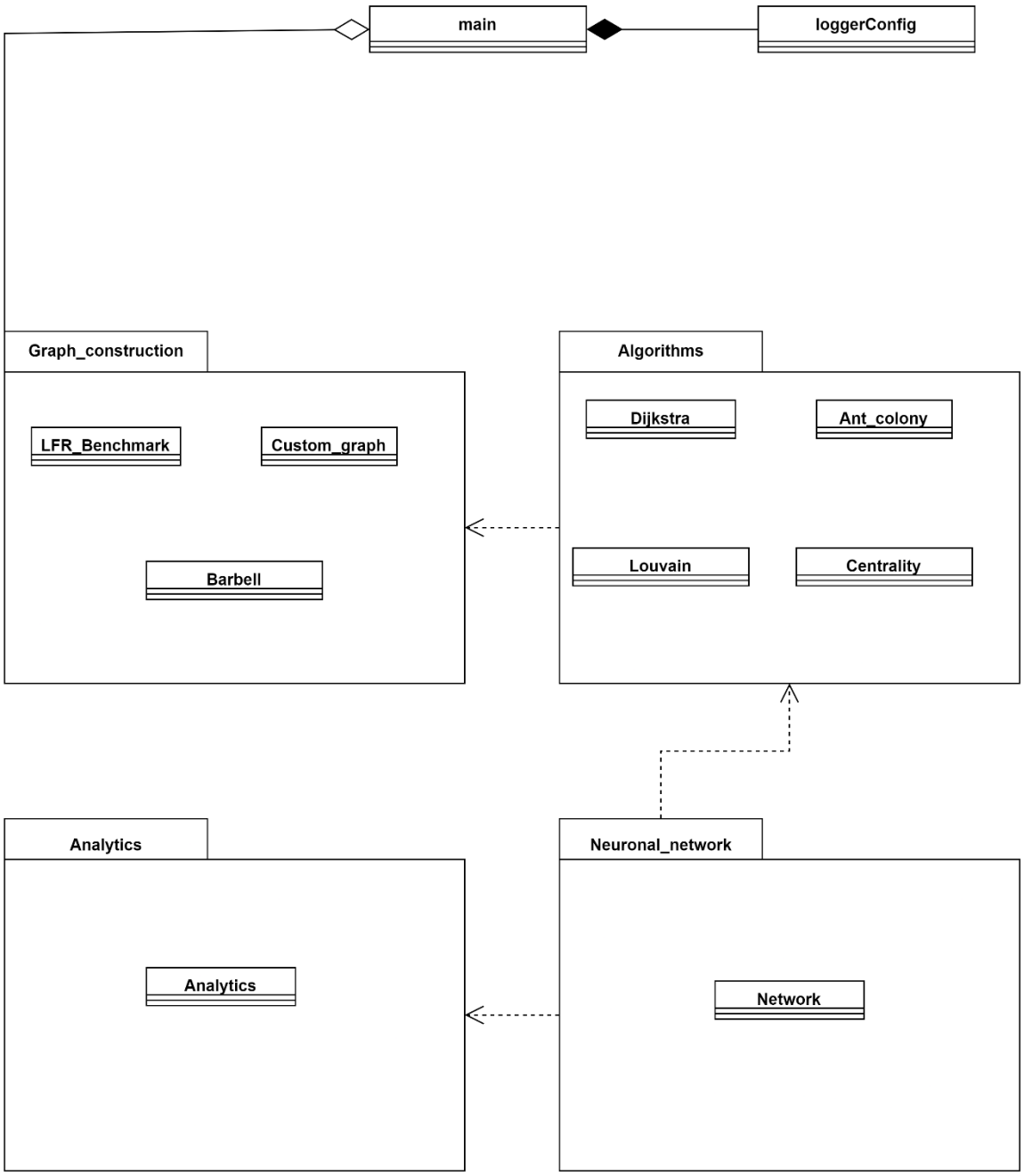


Figura 15: Diagrama UML de paquetes de la aplicación

4.3 Librerías utilizadas

La siguiente lista de librerías externas utilizadas durante el trabajo ha sido crucial para el desarrollo de este, sin ellas no se hubiera conseguido el objetivo con el rendimiento obtenido,

seguidamente se muestra las lista ordenada de mayor a menor importancia durante el proyecto en este proyecto es:

- **Deep Graph Library (DGL):** Deep Graph Library (DGL) es una biblioteca de Python diseñada para simplificar la implementación y el desarrollo de modelos de aprendizaje profundo para datos estructurados en forma de grafo. Proporciona una amplia gama de funcionalidades y herramientas específicamente diseñadas para redes neuronales de grafos (GNN, por sus siglas en inglés) cuya función principal en este trabajo ha sido la construcción de grafos; DGL permite a los usuarios crear y manipular estructuras de datos de grafos, tanto homogéneos como heterogéneos. Proporciona una API intuitiva para construir grafos a partir de diversas fuentes de datos, como matrices NumPy y grafos de NetworkX que han sido las más utilizadas en este proyecto.

  Además, ofrece una interfaz de alto nivel para definir y entrenar modelos GNN. Admite una variedad de arquitecturas de GNN, incluyendo redes convolucionales de grafos (GCN, por sus siglas en inglés) que ha sido la usada en este proyecto además de otras como GAT y GraphSAGE. Se pueden construir fácilmente modelos GNN apilando capas y definiendo funciones de propagación de mensajes.

  Propagación de mensajes: DGL utiliza un enfoque de propagación de mensajes para permitir que los nodos de un grafo interactúen y compartan información entre sí, esto ha sido muy beneficioso ya que permite una mejor detección de comunidades al compartir la información de las características. Esto facilita el procesamiento y la agregación de características en un grafo, lo que es fundamental en los modelos de aprendizaje profundo para grafos.

  En resumen, Deep Graph Library (DGL) es una biblioteca de Python que simplifica la implementación de modelos de aprendizaje profundo para datos estructurados en forma de grafo. Proporciona herramientas y funcionalidades específicas para redes neuronales de grafos que ha permitido una rápida evolución en este TFG.

- **TensorFlow:** DGL utiliza TensorFlow como backend para aprovechar las capacidades de cálculo en paralelo y aceleración de hardware proporcionadas por TensorFlow. Esto permite un procesamiento eficiente de grandes grafos y un mejor rendimiento en tareas como la detección de comunidades.

- **NetworkX:** Es una poderosa biblioteca de Python que proporciona herramientas para la creación, manipulación y análisis de redes complejas. Con su amplia gama de funciones y algoritmos, NetworkX se ha convertido en una herramienta esencial para estudiar y comprender las propiedades y estructuras de las redes.

- **MatplotLib:** Es una biblioteca de visualización en Python ampliamente utilizada que permite crear gráficos de alta calidad. Con su amplia gama de funciones y opciones de personalización, Matplotlib ofrece una forma flexible y eficiente de representar datos en forma de gráficos, histogramas, diagramas de dispersión y más. Su integración con NumPy y Pandas lo convierte en una herramienta poderosa para explorar y comunicar patrones y tendencias en los datos.

- **Pandas:** Es una biblioteca de Python ampliamente utilizada para el análisis y manipulación de datos. Proporciona estructuras de datos eficientes y flexibles, como DataFrames, que permiten organizar y procesar datos tabulares de manera intuitiva. Con Pandas, es posible realizar operaciones de filtrado, agregación, transformación y visualización de datos de forma sencilla. Además, su integración con otras bibliotecas, como NumPy y Matplotlib, lo convierte en una herramienta poderosa para el análisis exploratorio de datos

4.4 Problemas durante el desarrollo

Durante el desarrollo del proyecto hubo problemas con la librería de trabajo DGL, al empezar este trabajo en una fase poco estable del mismo, se empezó trabajando en PyTorch pero al cambiar la versión de la misma se hizo inutilizable debido a problemas de compatibilidad entre el backend que implementaba PyTorch y la versión actual de PyTorch que estaba usando el

sistema. Esta información al estar ofuscada dentro de la propia librería hizo imposible que se reconociera qué librería de PyTorch necesitaba en la versión actual la librería haciendo que se tuviera que descartar toda la parte asociada a la red neuronal.

También hubo problemas a la hora de implementar el algoritmo de Louvain por cuenta propia ya que en la implementación no se conseguía reducir la complejidad de $O(n^3)$ que al trabajar con redes de más de 1000 grafos hacía que el tiempo fuera demasiado grande como para que fuera viable realizar pruebas de experimentación con esta parte.

Hubo problemas al intentar alimentar a la red con las características ya que obligatoriamente tenían que estar en un tensor con un dtype=float64, esto provocó que se tuviera que limitar el tipo de datos que sirven para alimentar a la red con datos estrictamente numéricos.

5.Experimentación

El objetivo principal de esta sección es determinar la configuración óptima de los parámetros en los diferentes algoritmos y en la red neuronal propuesta, así como la evaluación de rendimiento de las diferentes partes que lo conforman.

El proyecto se ha implementado sobre la versión 3.10.4 de Python en el editor de código Visual Studio Code y los experimentos han sido realizados en un ASUS TUF GAMING que dispone de un procesador Intel Core I7 (3.3 GHz) y 16 GB de RAM.

Para la experimentación, se han generado 100 instancias (consultables en el repositorio de código en formato ". gml") variadas entre los grafos de Barbell y los grafos LFR_Benchmark con un número aleatorio de grafos dónde las 2 comunidades del Barbell estaban comprendidas entre los 15 y los 100 nodos mientras que en el LFR se ha contado con más de 500 nodos por las limitaciones en su generación que impone la librería de NetworkX.

5.1 Algoritmo ACO (Ant Colony Optimization)

Para la experimentación del algoritmo ACO (Ant Community Optimization), se han realizado 2 experimentaciones ya que la principal configuración de parámetros, por una parte, sería la configuración de la tasa de evaporación y por otra parte los parámetros alpha y rho, no se ha experimentado con el número de iteraciones o la cantidad de hormigas debido a que esos parámetros siempre suponen una mejora cuanto mayores sean a cambio de un coste computacional, es decir, siempre conseguiríamos mejores resultados a medida que aumentemos las iteraciones y el número de hormigas.

Para la experimentación con este algoritmo, se han usado las siguientes métricas:

- Tiempo (s): Tiempo de cómputo medio en segundos que ha requerido el algoritmo para encontrar la mejor solución
- Mejores: Número de soluciones mejores encontradas en el experimento para ese caso

- Desv (%): La desviación porcentual media con respecto al mejor resultado del experimento. Para ello se ha usado los valores obtenidos en la modularidad con la fórmula$\frac{M_a - M_b}{M_a}$
- Modularidad promedio: Valor medio de la modularidad obtenida

La tasa de evaporación desempeña un papel fundamental en el algoritmo de las hormigas, ya que influye en la capacidad de adaptación de las feromonas a medida que las hormigas exploran una variedad de caminos posibles. Toma valores entre 0 y 1 y cuando la tasa es próxima a 1, las feromonas depositadas por las hormigas se desvanecen rápidamente, lo que implica que las marcas antiguas se borren lo que favorece la exploración de nuevos caminos, mientras que, en contraste, si se establece una tasa de evaporación baja (próxima a cero) se fomentará el uso de caminos ya conocidos y consolidados, lo que puede ser beneficioso cuando se ha encontrado una solución prometedora. En conclusión, el ajuste de esta tasa permite el equilibrio entre los caminos ya conocidos y la búsqueda de nuevos caminos.

Evaporación	**Modularidad**	**Tiempo (s)**	**Mejor**	**Desv (%)**
0.25	**0.535926903**	**39.1898347**	**29**	**11.62%**
0.5	0.536301039	39.1668403	23	12.41%
0.75	0.533086687	39.1999375	20	12.05%
RND	0.532870502	39.1853035	28	19.00%

Tabla 1: Resultados experimento para la tasa de evaporación

Como se puede observar, aunque los resultados obtenidos son muy similares, configurando el parámetro a una tasa de evaporación del 0.25 se consigue obtener una mayor cantidad de veces el mejor resultado.

Los valores alpha y beta son parámetros que influyen en la elección de las aristas por parte de las hormigas, durante la construcción de las soluciones. Estos parámetros permiten ponderar la importancia relativa de la información de feromona y la información heurística en el proceso de decisión de las hormigas.

Alpha controla la influencia de la feromona, Un valor mayor de Alpha que de beta dará más peso a la feromona, lo que significa que las hormigas tendrán más probabilidades de elegir aristas con una alta cantidad de feromona depositada, ya que se considera que esas aristas representan caminos más prometedores basados en la experiencia acumulada.

Beta controla la información heurística en la decisión de las hormigas. Un valor mayor de beta que de Alpha dará más peso a la información heurística, lo que significa que las hormigas tendrán más probabilidades de elegir aristas cuya heurística sea mejor.

Para la experimentación con estos parámetros se ha decidido usar las métricas usadas anteriormente y se han planteado 3 situaciones que recopilan todos los casos posibles, Alpha > Beta, Alpha = Beta, Beta > Alpha.

Valores α y β (Alpha y Beta)	**Modularidad**	**Tiempo (s)**	**Mejor**	**Desv (%)**
$\alpha = \beta$	0.539537	39.218010	31	9.73%
$\alpha > \beta$	0.532676	39.217390	22	10.78%
$\alpha < \beta$	**0.553700**	**39.231180**	**47**	**7.66%**

Tabla 2: Resultados experimento parámetros alpha y beta

Se puede observar que hay una mejora de tiempo de un 0.02 de modularidad que, aunque sea una diferencia sutil simboliza que es la mejor configuración de este parámetro ya que ha obtenido también el mejor resultado en la mayoría de las pruebas.

Juntando los 2 experimentos se puede observar que la mejor configuración posible es una tasa de evaporación baja en torno al 0.25 y un Beta > Alpha.

En conclusión, el uso de este algoritmo en el contexto del problema de detección de comunidades para abordarlo de la mejor manera posible se debe dar más relevancia a mantener las particiones encontradas utilizando una tasa de evaporación baja y se observa también que

es más relevante la heurística usada para el algoritmo ante las feromonas depositadas en las aristas.

5.2 Algoritmo de Louvain

Para la experimentación del algoritmo de Louvain al no tener parámetros configurables simplemente se ha procedido a comparar la eficiencia de este algoritmo respecto al algoritmo de hormigas con su configuración de parámetros de mayor efectividad explicada anteriormente.

Para ello se ha vuelto a ejecutar el algoritmo sobre las 100 instancias obteniendo los siguientes y se han usado las métricas explicadas anteriormente obteniendo el siguiente resultado:

Algorithm	**AVG. Modularity**	**#Best**	**% Dev**
ACO	**0.553700405**	**66**	**1.29%**
Louvain	0.510307177	34	7.88%

Tabla 3: Comparativa ACO vs Louvain

5.3 Red Neuronal

Para la experimentación de la red neuronal se han fijado parámetros como puede ser el número de épocas de entrenamiento a 100 ya que se considera que deberían ser las apropiadas para encontrar un resultado óptimo. El parámetro con el que se ha experimentado en esta prueba es el "learning rate" o tasa de aprendizaje, este parámetro sirve para el ajuste en los pesos en el modelo de red neuronal en cada iteración. Si se usa un learning rate elevado ayudaría a que el modelo se ajuste más rápidamente a los datos proporcionados, pero puede causar que no converja nunca y que oscile entre 2 ajustes constantemente, por otro lado, un learning rate demasiado bajo podría causar que el entrenamiento sea más lento y que el modelo nunca se ajuste correctamente a los datos.

Para el experimento se han realizado 4 pruebas ajustando el valor del learning rate entre 0.025 y 0.1 ya que son los valores más recomendados para esta tarea (Géron), por ello se han realizado 4 pruebas sobre 100 instancias donde se probarán los valores 0.025, 0.050, 0.075, 0.1 donde se estudiarán las siguientes métricas:

- Tiempo (s): Tiempo de cómputo medio en segundos que ha requerido el algoritmo para encontrar la mejor solución
- Mejores: Número de soluciones mejores encontradas en el experimento para ese caso
- Desv (%): La desviación porcentual media con respecto al mejor resultado del experimento. Para ello se ha usado los valores obtenidos en NMI.
- NMI promedio: Valor medio del NMI obtenido.

Los resultados obtenidos han sido los siguientes:

Learning Rate	**NMI**	**Tiempo (s)**	**Mejor**	**Desv (%)**
0.025	**0.637785**	**44.74745**	**72**	**17.56%**
0.050	0.507636	44.54508	10	38.12%
0.075	0.469803	44.66482	10	42.90%
0.100	0.467214	44.67881	8	44.73%

Tabla 4: Resultados para los diferentes valores de learning rate

Como se puede observar, la mayor tasa de aciertos la posee el valor más bajo de learning rate, sin embargo, se puede observar que las instancias con mayor tamaño han sido las que ha obtenido mejor resultado el learning rate de 0.1 lo que puede significar que se obtendrá un mejor resultado con un learning rate más elevado en función de la medida del grafo.

6.Conclusiones y trabajos futuros

En este capítulo se comentarán las conclusiones, los objetivos que se han logrado y todo lo aprendido a lo largo del desarrollo de este proyecto además de los puntos débiles y los futuros trabajos y mejoras que se podrían realizar en este proyecto.

6.1 Conclusiones

Durante el desarrollo de este trabajo se ha logrado mejorar habilidades como la habilidad para trabajar con Python y sus diferentes librerías, el trabajo con algoritmos complejos que no se ven en la carrera, el trabajo con redes neuronales complejas y su tratamiento para poder ser usadas en grafos, el manejo de Git al subir el código, la familiaridad usando el editor de código Visual Studio Code. He considerado este trabajo muy enriquecedor puesto que me ha permitido seguir desarrollando mis habilidades en temas que me hubiese gustado que se vieran más en profundidad en la carrera y considero que puede ser muy beneficioso para mi futuro ya que tanto los conocimientos aprendidos redes neuronales como en algoritmia pueden ser muy útiles en el ámbito laboral.

6.2 Objetivos logrados

En este apartado se muestran los objetivos conseguidos tanto el principal como los secundarios que fueron planteados en el capítulo 2.

El objetivo principal que era resolver el problema de detección de comunidades usando un enfoque mixto entre el uso de algoritmos y el uso de una red neuronal puede decirse que se ha conseguido puesto que la red neuronal consigue detectar de media el 60% de las comunidades de un grafo, este resultado, aunque sea muy mejorable, demuestra que el objetivo era posible y que simplemente en futuros trabajos habría que hacer una mejor optimización de la red neuronal usada.

A continuación, se analizará punto por punto los objetivos planteados y su resolución:

- **Adquirir conocimientos sobre el desarrollo de redes neuronales y Deep Learning**: Considero que este objetivo ha sido logrado con éxito puesto que se ha conseguido obtener un mayor entendimiento tanto teórico como práctico sobre cómo funciona una red neuronal y cómo trabajar con ella además de ajustarla.
- **Adquirir conocimientos sobre algoritmos complejos**: El desarrollo de algoritmos como el de Louvain, ACO y el de centralidad de los nodos me ha ayudado a progresar a la hora de desarrollar algoritmos en base a su pseudocódigo o a su definición formal.
- **Mejorar las habilidades de programación adquiridas en otras asignaturas cursadas**: Considero que tras terminar el proyecto se ha conseguido mejor estas habilidades y programar con una mayor soltura
- **Profundizar el conocimiento en el lenguaje de programación Python**: Se ha conseguido pasar de un punto básico donde el alumno tenía que consultar constantemente operaciones básicas como crear una clase o la importación de librerías, a un punto donde se hace con soltura y normalidad

6.3 Puntos débiles

Algunos de los principales puntos débiles del proyecto son los siguiente:

1. Al utilizar un subgrafo para la detección de comunidades inicial usando un algoritmo, cabe la posibilidad de que algunas comunidades queden sin reconocerse puesto que no están en esta clasificación inicial haciendo imposible para la red clasificarlas más tarde.
2. Se necesitan grafos de un tamaño considerable para que la red neuronal tenga una entrada lo suficientemente grande y su porcentaje de éxito sea mayor, al mismo tiempo, trabajar con grafos grandes significa un mayor tiempo para que los algoritmos hagan la clasificación de la red, provocando así que siempre haya un mejor uso de uno de los componentes en detrimento del otro.

3. Se carece de interfaz gráfica y la traza no es lo suficientemente consistente para saber en todo momento si la clasificación de las comunidades se está realizando correctamente.
4. El acierto de la red neuronal a pesar de ser alto (en torno al 60%) sigue siendo muy bajo para que se utilice en un entorno realista.

6.4 Futuros trabajos y mejoras

Tras finalizar este trabajo se va a intentar realizar las siguientes mejoras sobre el proyecto realizado:

- Optimización de los algoritmos ya implementados
- Implementación de los algoritmos y enfoques vistos en el estado del arte que se han quedado fuera del alcance del proyecto
- Mejoras en la red neuronal para conseguir un mayor porcentaje de éxito en la clasificación.
- Mejora de la red neuronal a una red LGNN para un mayor optimización en la tarea de clasificación

Referencias

Ant Colony Optimization. (14 de 04 de 2023). Obtenido de https://es.wikipedia.org/wiki/Algoritmo_de_la_colonia_de_hormigas

Barbell Graph. (s.f.). Obtenido de https://networkx.org/documentation/stable/reference/generated/networkx.generators.classic.barbell_graph.html

Chollet, F. (s.f.). *Deep Learning with Python.*

Eric Bonabeau, M. D. (s.f.). *Swarm Intelligence: From Natural to Arificial Systems.*

Faust, S. W. (s.f.). *Social Network Analysis.*

Friedman, D. K. (s.f.). *Probabilistic Graphical Models: Principles and Techniques.*

Géron, A. (s.f.). *Hans On machine Learning with Scikit-Learn, Keas, and TensorFlow.*

Han, W. L. (s.f.). *Graph Representation Learning.*

Ian Goodfellow, Y. B. (s.f.). *Deep Learning.*

LFR_benchmark_graph. (s.f.). Obtenido de https://networkx.org/documentation/stable/reference/generated/networkx.generators.community.LFR_benchmark_graph.html

Liu, L. T. (s.f.). *Community Detection and Mining in Social Media.*

Luxburg, U. v. (s.f.). *Spectral Clustering for Graphs.*

Newman, M. (s.f.). *Graph Representation Learning.*

Newman, M. (s.f.). *Networks: An Introduction.*

Sergio Pérez-Peló, J. S.-O.-P. (s.f.). A fast variable neighborhood search approach for multi-objective communitiy detection.

Stüzle, M. D. (s.f.). *Ant Colony Optimization.*

Subramanian, V. (s.f.). *PyTorch for Deep Learning.*

William L. Hamilton, R. Y. (s.f.). *Deep Learning on Graphs.*

Apéndices

A. Resultados por cada instancia para la tasa de evaporación (rho)

1.Siendo la tasa de evaporación = 0.25

Instancia	Modularidad1	Time1	Best1	Dev1
graph0.gml	0.54354296	38.05191	0	0.125992
graph1.gml	0.5	38.8798	0	0.280498
graph2.gml	0.53131485	38.88892	1	0
graph3.gml	0.57717421	39.42492	1	0
graph4.gml	0.53026862	39.57883	0	0.047847
graph5.gml	0.51309813	39.65522	0	0.203482
graph6.gml	0.46593312	39.29783	0	0.21748
graph7.gml	0.50110497	39.44945	0	0.138669
graph8.gml	0.48380844	38.8864	0	0.202252
graph9.gml	0.51752291	39.44554	0	0.132235
graph10.gml	0.51106195	39.4127	0	0.128477
graph11.gml	0.62410917	39.53751	1	0
graph12.gml	0.46814551	39.32686	0	0.151159
graph13.gml	0.54336675	39.62349	0	0.235948
graph14.gml	0.47495888	39.28373	0	0.224109
graph15.gml	0.5	39.55389	0	0.190118
graph16.gml	0.54416947	39.05504	0	0.057639
graph17.gml	0.57948939	39.26007	0	0.041981
graph18.gml	0.48654946	38.95085	0	0.284584
graph19.gml	0.5998512	39.27212	1	0
graph20.gml	0.53982301	39.12835	0	0.038094
graph21.gml	0.48547141	39.356	0	0.052541
graph22.gml	0.48876185	39.02263	0	0.272927
graph23.gml	0.59160858	39.38217	1	0
graph24.gml	0.51903046	38.97391	0	0.055104
graph25.gml	0.53248101	39.31647	0	0.176921
graph26.gml	0.59923325	39.40277	1	0
graph27.gml	0.47407784	39.39238	0	0.225549
graph28.gml	0.51638612	39.37295	0	0.014831
graph29.gml	0.46372073	39.22087	0	0.077972
graph30.gml	0.59824575	39.42817	1	0
graph31.gml	0.57618916	39.11785	1	0
graph32.gml	0.45124912	39.23749	0	0.160976
graph33.gml	0.49778761	38.8828	0	0.008811
graph34.gml	0.5	37.28026	0	0.057082

graph35.gml	0.5	38.79441	0	0.089749
graph36.gml	0.51417373	39.34297	0	0.085746
graph37.gml	0.54957319	39.1619	0	0.015053
graph38.gml	0.4753113	39.00949	0	0.099885
graph39.gml	0.54638186	39.01705	0	0.045458
graph40.gml	0.46391652	39.97286	0	0.259392
graph41.gml	0.53743441	38.88106	0	0.068068
graph42.gml	0.70191479	39.2546	1	0
graph43.gml	0.49097423	39.14697	0	0.271123
graph44.gml	0.5197353	38.90823	0	0.12528
graph45.gml	0.50221239	39.27888	1	0
graph46.gml	0.48433707	39.23257	0	0.107479
graph47.gml	0.49778761	39.37019	0	0.128354
graph48.gml	0.63675699	38.58703	0	0.025732
graph49.gml	0.5	39.18007	0	0.150404
graph50.gml	0.54708669	39.1715	0	0.116287
graph51.gml	0.5871838	39.33915	1	0
graph52.gml	0.5	39.5582	0	0.136238
graph53.gml	0.48876185	40.93377	0	0.140963
graph54.gml	0.61968439	39.33002	1	0
graph55.gml	0.57514292	39.42498	1	0
graph56.gml	0.70948072	39.2329	1	0
graph57.gml	0.50442478	39.67141	0	0.058643
graph58.gml	0.50407236	39.29302	0	0.105545
graph59.gml	0.55142092	39.60978	1	0
graph60.gml	0.55329191	39.20854	0	0.07183
graph61.gml	0.62099616	39.31751	1	0
graph62.gml	0.61435899	39.20564	0	0.135361
graph63.gml	0.50442478	38.88281	0	0.175656
graph64.gml	0.6303046	39.0084	1	0
graph65.gml	0.45831579	39.19153	0	0.193074
graph66.gml	0.5448743	39.24917	1	0
graph67.gml	0.58187799	39.08887	1	0
graph68.gml	0.70819955	38.87791	1	0
graph69.gml	0.73300572	39.15034	1	0
graph70.gml	0.53248101	39.12601	1	0
graph71.gml	0.49539901	39.11708	0	0.009202
graph72.gml	0.49992169	38.99392	0	0.144102
graph73.gml	0.51531052	38.99724	0	0.238447
graph74.gml	0.56355235	39.19719	0	0.125543
graph75.gml	0.49668019	39.17226	0	0.151044
graph76.gml	0.5	39.26573	0	0.112771
graph77.gml	0.52765364	38.91011	1	0
graph78.gml	0.55261644	38.8945	0	0.069389
graph79.gml	0.52910246	39.12533	1	0

graph80.gml	0.51531052	39.21696	1	0
graph81.gml	0.5	38.48602	0	0.161121
graph82.gml	0.59160858	39.42767	0	0.058484
graph83.gml	0.50185997	39.16028	0	0.300233
graph84.gml	0.5	39.47122	0	0.216083
graph85.gml	0.58720338	39.23009	0	0.062147
graph86.gml	0.46283969	39.43012	0	0.153278
graph87.gml	0.50203618	39.03387	0	0.082804
graph88.gml	0.50110497	39.30354	0	0.234374
graph89.gml	0.5	39.24079	0	0.191322
graph90.gml	0.47186546	39.24625	0	0.147833
graph91.gml	0.48654946	38.91229	0	0.289605
graph92.gml	0.48654946	38.59585	0	0.23323
graph93.gml	0.48654946	39.33107	0	0.082447
graph94.gml	0.6774953	39.20065	1	0
graph95.gml	0.5	39.13469	0	0.267202
graph96.gml	0.74037586	39.21016	1	0
graph97.gml	0.49778761	38.87211	0	0.269816
graph98.gml	0.56950427	39.18785	1	0
graph99.gml	0.51437931	39.25872	1	0

Tabla 5: Resultados por cada instancia para la tasa de evaporación = 0.25

2. Siendo la tasa de evaporación = 0.5

Instancia	Modularidad2	Time2	Best2	Dev2
graph0.gml	0.621896781	39.31996	1	0
graph1.gml	0.579665596	39.31215	0	0.165859
graph2.gml	0.513098128	39.17835	0	0.034286
graph3.gml	0.505353542	39.67693	0	0.124435
graph4.gml	0.556915185	39.65436	1	0
graph5.gml	0.5	39.69979	0	0.223815
graph6.gml	0.531314854	39.32012	0	0.107673
graph7.gml	0.502231968	39.38941	0	0.136732
graph8.gml	0.489896184	39.39853	0	0.192214
graph9.gml	0.596385778	39.19876	1	0
graph10.gml	0.579665596	39.30884	0	0.011485
graph11.gml	0.483386277	39.2966	0	0.225478
graph12.gml	0.551511473	39.28869	1	0
graph13.gml	0.598245751	39.25046	0	0.15878
graph14.gml	0.612146605	38.87943	1	0
graph15.gml	0.545891172	39.23364	0	0.115785
graph16.gml	0.577453207	39.12882	1	0
graph17.gml	0.5	39.27809	0	0.173394
graph18.gml	0.515681289	39.5566	0	0.241749
graph19.gml	0.5	39.23839	0	0.16646
graph20.gml	0.488761845	39.1604	0	0.12908
graph21.gml	0.495222805	39.41577	0	0.03351
graph22.gml	0.488233221	39.34723	0	0.273714
graph23.gml	0.573909468	39.18571	0	0.029917
graph24.gml	0.486549456	39.34439	0	0.114236
graph25.gml	0.646937896	39.18242	1	0
graph26.gml	0.576650482	39.2287	0	0.037686
graph27.gml	0.612146605	39.28019	1	0
graph28.gml	0.510885739	39.15224	0	0.025325
graph29.gml	0.475311301	39.14262	0	0.054926
graph30.gml	0.58328765	39.20307	0	0.025003
graph31.gml	0.515310518	39.05261	0	0.105657
graph32.gml	0.537825985	38.6571	1	0
graph33.gml	0.5	39.04504	0	0.004405
graph34.gml	0.496906571	38.1804	0	0.062915
graph35.gml	0.542288697	39.09771	0	0.012762
graph36.gml	0.530268619	39.07825	0	0.057128
graph37.gml	0.488384956	39.12727	0	0.124715
graph38.gml	0.51786431	39.1046	0	0.019301
graph39.gml	0.504424779	39.1588	0	0.118758
graph40.gml	0.476995066	39.12446	0	0.238513

graph41.gml	0.576688415	39.10641	1	0
graph42.gml	0.515310518	39.24198	0	0.26585
graph43.gml	0.5	39.40611	0	0.257724
graph44.gml	0.530268619	39.11982	0	0.107552
graph45.gml	0.498687006	39.51285	0	0.00702
graph46.gml	0.5	38.9801	0	0.078616
graph47.gml	0.571088917	39.4742	1	0
graph48.gml	0.488761845	39.01257	0	0.252172
graph49.gml	0.588515154	39.53445	1	0
graph50.gml	0.569306034	39.21512	0	0.080396
graph51.gml	0.497787611	39.46831	0	0.152246
graph52.gml	0.5	39.06783	0	0.136238
graph53.gml	0.489055525	39.67369	0	0.140447
graph54.gml	0.551687681	39.24458	0	0.109728
graph55.gml	0.514203099	39.1783	0	0.105956
graph56.gml	0.526548673	39.17031	0	0.257839
graph57.gml	0.5	39.32235	0	0.066901
graph58.gml	0.5	38.93903	0	0.112771
graph59.gml	0.489896184	39.14756	0	0.111575
graph60.gml	0.486549456	39.15168	0	0.183793
graph61.gml	0.536905787	39.18083	0	0.135412
graph62.gml	0.674503436	38.77745	0	0.050714
graph63.gml	0.524336283	39.10482	0	0.143116
graph64.gml	0.53539823	38.98948	0	0.150572
graph65.gml	0.560184823	39.26207	0	0.013719
graph66.gml	0.495575221	39.27175	0	0.090478
graph67.gml	0.485442037	39.083	0	0.165732
graph68.gml	0.5	39.28765	0	0.293984
graph69.gml	0.579567703	39.13463	0	0.209327
graph70.gml	0.470357898	39.19123	0	0.116667
graph71.gml	0.5	39.09793	1	0
graph72.gml	0.497905083	39.17273	0	0.147555
graph73.gml	0.676657089	39.12486	1	0
graph74.gml	0.582150863	39.28025	0	0.096684
graph75.gml	0.5	39.03396	0	0.14537
graph76.gml	0.563552353	39.43817	1	0
graph77.gml	0.519030464	38.95557	0	0.016342
graph78.gml	0.511061947	39.37701	0	0.139367
graph79.gml	0.502212389	39.01603	0	0.050822
graph80.gml	0.498892582	38.80156	0	0.03186
graph81.gml	0.596033362	39.07588	1	0
graph82.gml	0.628357741	38.76444	1	0
graph83.gml	0.458414911	38.87663	0	0.36081
graph84.gml	0.637822803	39.16683	1	0
graph85.gml	0.562397212	39.12958	0	0.101767

graph86.gml	0.526372465	39.04188	0	0.037051
graph87.gml	0.520076699	38.5297	0	0.049845
graph88.gml	0.654503828	39.12972	1	0
graph89.gml	0.509102857	39.231	0	0.176599
graph90.gml	0.553723862	39.01948	1	0
graph91.gml	0.493186624	38.9907	0	0.279914
graph92.gml	0.514173731	39.00876	0	0.189696
graph93.gml	0.530268619	39.24401	1	0
graph94.gml	0.477171274	39.09248	0	0.295683
graph95.gml	0.682316548	38.93834	1	0
graph96.gml	0.568427441	39.03897	0	0.232245
graph97.gml	0.681729188	39.09009	1	0
graph98.gml	0.521242854	38.83495	0	0.084743
graph99.gml	0.497787611	39.05542	0	0.032256

Tabla 6:Resultados por cada instancia para la tasa de evaporación = 0.5

3. Siendo la tasa de evaporación = 0.75

Instancia	Modularidad	Time	Best	Dev
graph0.gml	0.481948469	40.05006	0	0.225035
graph1.gml	0.497787611	38.90214	0	0.283682
graph2.gml	0.515310518	39.61137	0	0.030122
graph3.gml	0.515310518	39.33842	0	0.107184
graph4.gml	0.540273318	39.56812	0	0.029882
graph5.gml	0.644176081	39.69622	1	0
graph6.gml	0.595426423	39.4282	1	0
graph7.gml	0.489896184	39.28632	0	0.157936
graph8.gml	0.5	39.24574	0	0.175554
graph9.gml	0.5	39.4571	0	0.161616
graph10.gml	0.475311301	39.71326	0	0.189443
graph11.gml	0.504072363	39.37617	0	0.192333
graph12.gml	0.50331736	39.24324	0	0.087386
graph13.gml	0.549299084	39.29261	0	0.227606
graph14.gml	0.515310518	39.05557	0	0.158191
graph15.gml	0.487683795	39.29657	0	0.210068
graph16.gml	0.495496907	39.40018	0	0.141927
graph17.gml	0.549299084	39.46965	0	0.091892
graph18.gml	0.553331065	39.34841	0	0.186389
graph19.gml	0.556738977	39.20366	0	0.071872
graph20.gml	0.517696668	39.11774	0	0.077521
graph21.gml	0.512393296	39.15972	1	0
graph22.gml	0.672232311	39.10217	1	0
graph23.gml	0.484337066	39.07054	0	0.181322
graph24.gml	0.5	39.17871	0	0.089749
graph25.gml	0.46372073	39.17966	0	0.283207
graph26.gml	0.561339964	38.89728	0	0.063236
graph27.gml	0.5	39.16568	0	0.183202
graph28.gml	0.504424779	39.08172	0	0.037651
graph29.gml	0.502935576	39.21522	1	0
graph30.gml	0.458414911	39.09161	0	0.233735
graph31.gml	0.531314854	39.36977	0	0.077881
graph32.gml	0.521242854	39.17058	0	0.030834
graph33.gml	0.502212389	39.32555	1	0
graph34.gml	0.473098911	39.64725	0	0.107813
graph35.gml	0.484337066	39.39731	0	0.118263
graph36.gml	0.562397212	38.80435	1	0
graph37.gml	0.557972433	39.39856	1	0
graph38.gml	0.5	39.11543	0	0.053131
graph39.gml	0.514173731	39.23001	0	0.101726
graph40.gml	0.486549456	39.23579	0	0.223261

graph41.gml	0.5	39.39541	0	0.132981
graph42.gml	0.64638847	39.2012	0	0.079107
graph43.gml	0.673604041	39.25413	1	0
graph44.gml	0.594173389	39.17779	1	0
graph45.gml	0.5	39.17336	0	0.004405
graph46.gml	0.542661916	38.68868	1	0
graph47.gml	0.560184823	39.1979	0	0.019094
graph48.gml	0.653575065	39.16534	1	0
graph49.gml	0.549122876	39.29187	0	0.066935
graph50.gml	0.619077453	39.21142	1	0
graph51.gml	0.500626517	39.23146	0	0.147411
graph52.gml	0.50538291	38.93126	0	0.126938
graph53.gml	0.568964631	39.16998	1	0
graph54.gml	0.506284752	39.32128	0	0.182996
graph55.gml	0.475917016	39.18556	0	0.172524
graph56.gml	0.582836117	39.20712	0	0.178503
graph57.gml	0.495399013	39.09545	0	0.075487
graph58.gml	0.483875744	38.59408	0	0.141383
graph59.gml	0.497787611	38.90345	0	0.097264
graph60.gml	0.531314854	39.23605	0	0.108697
graph61.gml	0.545205919	39.03792	0	0.122046
graph62.gml	0.710537973	38.98465	1	0
graph63.gml	0.530621035	39.22536	0	0.132845
graph64.gml	0.5	39.36028	0	0.206733
graph65.gml	0.5	39.09996	0	0.119683
graph66.gml	0.529661681	39.10491	0	0.02792
graph67.gml	0.517522907	38.93035	0	0.110599
graph68.gml	0.503141152	39.14026	0	0.289549
graph69.gml	0.641377555	39.11616	0	0.125003
graph70.gml	0.519030464	39.28082	0	0.02526
graph71.gml	0.474958885	39.22364	0	0.050082
graph72.gml	0.550003916	39.0147	0	0.058358
graph73.gml	0.629922811	39.14116	0	0.069066
graph74.gml	0.563552353	39.08062	0	0.125543
graph75.gml	0.563552353	39.09928	0	0.036743
graph76.gml	0.482927402	39.20981	0	0.143066
graph77.gml	0.517522907	39.04414	0	0.0192
graph78.gml	0.513098128	39.47773	0	0.135938
graph79.gml	0.52805623	39.1128	0	0.001977
graph80.gml	0.467264469	39.1597	0	0.093237
graph81.gml	0.486549456	39.15929	0	0.183688
graph82.gml	0.5	39.1741	0	0.204275
graph83.gml	0.717181259	39.14644	1	0
graph84.gml	0.477171274	39.30768	0	0.251875
graph85.gml	0.62611476	39.28969	1	0

graph86.gml	0.546625372	38.88468	1	0
graph87.gml	0.547359572	39.31788	1	0
graph88.gml	0.5	39.12144	0	0.236063
graph89.gml	0.618293083	39.47387	1	0
graph90.gml	0.513274336	39.11217	0	0.07305
graph91.gml	0.513098128	39.22252	0	0.250842
graph92.gml	0.474077845	39.0458	0	0.252885
graph93.gml	0.5	39.04364	0	0.057082
graph94.gml	0.567838858	39.07348	0	0.161856
graph95.gml	0.547086694	39.2254	0	0.198192
graph96.gml	0.52805623	38.74878	0	0.286773
graph97.gml	0.549299084	39.04064	0	0.194256
graph98.gml	0.565764743	39.02899	0	0.006566
graph99.gml	0.474958885	39.16255	0	0.076637

Tabla 7:Resultados por cada instancia para la tasa de evaporación = 0.75

4. Siendo la tasa de evaporación = RND

Instancia	Modularidad	Time	Best	Dev
graph0.gml	0.5	39.20081	0	0.196008
graph1.gml	0.694925209	39.11086	1	0
graph2.gml	0.514379307	39.30195	0	0.031875
graph3.gml	0.5	39.50561	0	0.13371
graph4.gml	0.5	39.59834	0	0.102197
graph5.gml	0.488233221	39.64929	0	0.242081
graph6.gml	0.483906336	39.67278	0	0.187294
graph7.gml	0.581780092	39.58152	1	0
graph8.gml	0.606467568	39.45715	1	0
graph9.gml	0.532440628	39.29441	0	0.107221
graph10.gml	0.586400658	39.3536	1	0
graph11.gml	0.48922072	39.33138	0	0.21613
graph12.gml	0.46719472	38.93117	0	0.152883
graph13.gml	0.71116449	38.98948	1	0
graph14.gml	0.541330566	39.21825	0	0.115685
graph15.gml	0.617374109	39.22597	1	0
graph16.gml	0.577355314	39.187	0	0.00017
graph17.gml	0.60488292	39.13419	1	0
graph18.gml	0.680093146	39.04233	1	0
graph19.gml	0.501507557	39.09249	0	0.163947
graph20.gml	0.56120169	39.22908	1	0
graph21.gml	0.5	38.88181	0	0.024187
graph22.gml	0.5	39.20722	0	0.25621
graph23.gml	0.550404055	39.40729	0	0.069648
graph24.gml	0.549299084	39.36986	1	0
graph25.gml	0.487654427	39.09443	0	0.246211
graph26.gml	0.44770538	39.0144	0	0.25287
graph27.gml	0.572930535	39.5253	0	0.064063
graph28.gml	0.524160075	39.17379	1	0
graph29.gml	0.5	39.49415	0	0.005837
graph30.gml	0.523082025	39.46028	0	0.12564
graph31.gml	0.560184823	39.19533	0	0.027776
graph32.gml	0.474958885	39.33467	0	0.116891
graph33.gml	0.484337066	39.14351	0	0.035593
graph34.gml	0.530268619	39.27422	1	0
graph35.gml	0.549299084	39.07219	1	0
graph36.gml	0.497787611	39.02598	0	0.114883
graph37.gml	0.5	39.00124	0	0.103898
graph38.gml	0.52805623	39.13509	1	0
graph39.gml	0.572401911	38.96504	1	0
graph40.gml	0.626399875	38.75185	1	0

graph41.gml	0.528584854	39.28763	0	0.083413
graph42.gml	0.510885739	39.27701	0	0.272154
graph43.gml	0.530268619	39.20165	0	0.212789
graph44.gml	0.497787611	39.12557	0	0.162218
graph45.gml	0.501859973	39.05737	0	0.000702
graph46.gml	0.498892582	39.09191	0	0.080657
graph47.gml	0.484337066	39.23366	0	0.151906
graph48.gml	0.513098128	39.35549	0	0.214936
graph49.gml	0.5	39.04476	0	0.150404
graph50.gml	0.481596053	39.30474	0	0.222075
graph51.gml	0.578862871	39.15549	0	0.014171
graph52.gml	0.578862871	39.38113	1	0
graph53.gml	0.497082779	39.58421	0	0.126338
graph54.gml	0.562749628	39.38515	0	0.091877
graph55.gml	0.488761845	39.03381	0	0.150191
graph56.gml	0.488761845	39.29016	0	0.311099
graph57.gml	0.535848539	39.21579	1	0
graph58.gml	0.563552353	38.87272	1	0
graph59.gml	0.498414128	38.92535	0	0.096128
graph60.gml	0.596110453	39.15916	1	0
graph61.gml	0.497787611	39.15439	0	0.198405
graph62.gml	0.474958885	39.03613	0	0.33155
graph63.gml	0.611910437	39.06744	1	0
graph64.gml	0.513274336	39.15015	0	0.185673
graph65.gml	0.567977132	39.1052	1	0
graph66.gml	0.519789138	39.1773	0	0.046038
graph67.gml	0.516444857	39.29456	0	0.112452
graph68.gml	0.497787611	39.18572	0	0.297108
graph69.gml	0.641181768	38.54095	0	0.12527
graph70.gml	0.466207221	39.08512	0	0.124462
graph71.gml	0.484854677	39.36918	0	0.030291
graph72.gml	0.584090375	39.09102	1	0
graph73.gml	0.5	39.23235	0	0.261073
graph74.gml	0.644459971	39.20979	1	0
graph75.gml	0.585048506	39.38935	1	0
graph76.gml	0.486549456	39.6375	0	0.136638
graph77.gml	0.495399013	39.26284	0	0.061128
graph78.gml	0.593820973	39.24948	1	0
graph79.gml	0.5	39.4162	0	0.055003
graph80.gml	0.469653066	39.08738	0	0.088602
graph81.gml	0.510885739	39.39586	0	0.142857
graph82.gml	0.53358598	39.27772	0	0.150825
graph83.gml	0.525843841	38.88144	0	0.266791
graph84.gml	0.50646096	39.27558	0	0.205954
graph85.gml	0.466186418	38.80905	0	0.25543

graph86.gml	0.501859973	39.05461	0	0.081894
graph87.gml	0.5	38.93794	0	0.086524
graph88.gml	0.486549456	39.03224	0	0.256613
graph89.gml	0.618293083	39.47387	1	0
graph90.gml	0.513274336	39.11217	0	0.07305
graph91.gml	0.513098128	39.22252	0	0.250842
graph92.gml	0.474077845	39.0458	0	0.252885
graph93.gml	0.5	39.04364	0	0.057082
graph94.gml	0.567838858	39.07348	0	0.161856
graph95.gml	0.547086694	39.2254	0	0.198192
graph96.gml	0.52805623	38.74878	0	0.286773
graph97.gml	0.549299084	39.04064	0	0.194256
graph98.gml	0.565764743	39.02899	0	0.006566
graph99.gml	0.474958885	39.16255	0	0.076637

Tabla 8: Resultados por cada instancia para la tasa de evaporación = RND

B. Resultados para cada instancia para los parámetros Alpha y Beta

1.Siendo Alpha = Beta

Instancia	Modularidad	Time	Best	Dev
graph0.gml	0.543366748	39.43952	0	0.138266
graph1.gml	0.473704626	39.19296	0	0.165979
graph2.gml	0.586400658	38.93026	0	0.231829
graph3.gml	0.520869635	39.32974	0	0.07384
graph4.gml	0.484337066	39.50813	0	0.253292
graph5.gml	0.513098128	39.5704	0	0.089529
graph6.gml	0.666035663	39.59814	1	0
graph7.gml	0.5	39.18915	0	0.10966
graph8.gml	0.606214269	39.32563	0	0.208269
graph9.gml	0.632665048	39.21897	1	0
graph10.gml	0.5	39.16126	0	0.160981
graph11.gml	0.602992355	39.21594	1	0
graph12.gml	0.504424779	39.02957	0	0.244679
graph13.gml	0.635297155	39.27961	1	0
graph14.gml	0.5	38.80746	0	0.130651
graph15.gml	0.488761845	39.32533	0	0.285129
graph16.gml	0.591960999	39.21697	1	0
graph17.gml	0.5	39.33119	0	0.086068
graph18.gml	0.50646096	38.89839	0	0.197579
graph19.gml	0.502036181	39.33524	0	0.040919
graph20.gml	0.526196257	39.54968	1	0
graph21.gml	0.479383664	39.21875	0	0.244523
graph22.gml	0.492481792	39.16383	0	0.335077
graph23.gml	0.578862871	39.07528	1	0
graph24.gml	0.561339964	39.03868	0	0.130418
graph25.gml	0.493186624	39.24037	0	0.088936
graph26.gml	0.528368265	39.14604	1	0
graph27.gml	0.5	39.31541	0	0.204275
graph28.gml	0.48956457	39.16485	0	0.096001
graph29.gml	0.5	39.29016	0	0.123099
graph30.gml	0.628433609	39.37279	1	0
graph31.gml	0.567174407	39.2052	0	0.039933
graph32.gml	0.546381862	39.24887	0	0.216787
graph33.gml	0.596207123	41.38868	1	0
graph34.gml	0.474958885	39.80209	0	0.128315

graph35.gml	0.5	39.26703	0	0.107437
graph36.gml	0.618371398	39.36303	1	0
graph37.gml	0.5	39.1536	0	0.025862
graph38.gml	0.5	39.30481	0	0.190247
graph39.gml	0.497787611	39.31947	0	0.007418
graph40.gml	0.5	39.47041	0	0.029711
graph41.gml	0.660917065	39.25593	1	0
graph42.gml	0.557041223	39.11722	0	0.172803
graph43.gml	0.573810351	39.37263	1	0
graph44.gml	0.497787611	39.18017	0	0.036822
graph45.gml	0.5	39.10701	1	0
graph46.gml	0.592763725	39.46415	0	0.15998
graph47.gml	0.491550581	39.06372	0	0.256512
graph48.gml	0.5	39.16934	0	0.033859
graph49.gml	0.5	39.05415	0	0.12159
graph50.gml	0.490269402	39.09853	0	0.019461
graph51.gml	0.5	39.17283	0	0.081976
graph52.gml	0.488233221	39.15642	0	0.201746
graph53.gml	0.607721826	38.80891	0	0.063989
graph54.gml	0.648786857	38.88252	1	0
graph55.gml	0.486549456	39.00602	0	0.063851
graph56.gml	0.509652283	39.20149	0	0.124125
graph57.gml	0.523983867	39.03722	0	0.130159
graph58.gml	0.754424779	39.2975	1	0
graph59.gml	0.685449135	39.30276	1	0
graph60.gml	0.550315951	39.24867	0	0.229687
graph61.gml	0.472746495	39.46603	0	0.244669
graph62.gml	0.54185919	38.9393	0	0.23608
graph63.gml	0.5	39.34325	0	0.150404
graph64.gml	0.545979276	39.15141	1	0
graph65.gml	0.499295168	39.37319	0	0.00141
graph66.gml	0.561260426	40.03119	1	0
graph67.gml	0.641632078	39.47316	1	0
graph68.gml	0.5	39.14044	0	0.147056
graph69.gml	0.473098911	39.22383	0	0.20919
graph70.gml	0.548103561	39.46018	0	0.156484
graph71.gml	0.449565354	39.21958	0	0.365365
graph72.gml	0.498510798	39.05002	0	0.115414
graph73.gml	0.597638813	39.04381	0	0.144093
graph74.gml	0.500400139	38.76821	0	0.270821
graph75.gml	0.601613282	39.12194	1	0
graph76.gml	0.581877986	39.1839	0	0.264211
graph77.gml	0.561339964	39.12679	1	0
graph78.gml	0.497787611	39.39748	0	0.089409
graph79.gml	0.486549456	39.11185	0	0.106334

graph80.gml	0.497787611	39.28814	1	0
graph81.gml	0.503141152	39.12756	0	0.008775
graph82.gml	0.618371398	39.23581	1	0
graph83.gml	0.635601848	38.7751	1	0
graph84.gml	0.514173731	39.2307	0	0.08762
graph85.gml	0.563552353	38.72712	1	0
graph86.gml	0.513274336	38.97545	0	0.090999
graph87.gml	0.51513431	39.02142	0	0.119502
graph88.gml	0.514203099	38.96096	0	0.09272
graph89.gml	0.496680192	38.75212	0	0.059418
graph90.gml	0.669433785	39.26443	1	0
graph91.gml	0.493010416	39.08985	0	0.123377
graph92.gml	0.499430995	38.91708	0	0.229291
graph93.gml	0.5	38.90665	0	0.006591
graph94.gml	0.617055956	39.22089	1	0
graph95.gml	0.539118177	39.17731	1	0
graph96.gml	0.5	39.18298	0	0.11778
graph97.gml	0.567977132	39.13677	1	0
graph98.gml	0.513098128	39.21895	0	0.206882
graph99.gml	0.566821991	38.9664	1	0

Tabla 9: Resultados para cada instancia Alpha = Beta

2.Siendo Alpha > Beta

Instancia	Modularidad	Time	Best	Dev
graph0.gml	0.630550552	40.69444	1	0
graph1.gml	0.567977132	39.25883	1	0
graph2.gml	0.488761845	39.14817	0	0.359733
graph3.gml	0.5	39.58517	0	0.110949
graph4.gml	0.648630228	39.69517	1	0
graph5.gml	0.563552353	39.79165	1	0
graph6.gml	0.433647897	39.18732	0	0.348912
graph7.gml	0.536905787	39.23283	0	0.043943
graph8.gml	0.5	38.79751	0	0.346988
graph9.gml	0.466333258	39.39007	0	0.262907
graph10.gml	0.569042946	39.25685	0	0.045125
graph11.gml	0.497787611	39.299	0	0.174471
graph12.gml	0.667828334	39.42423	1	0
graph13.gml	0.5	40.24126	0	0.212967
graph14.gml	0.575142924	39.23947	1	0
graph15.gml	0.683706633	39.28045	1	0
graph16.gml	0.464884437	39.2126	0	0.21467
graph17.gml	0.547086694	39.24748	1	0
graph18.gml	0.590825437	39.15368	0	0.063914
graph19.gml	0.523455243	39.05851	1	0
graph20.gml	0.513098128	39.1648	0	0.024892
graph21.gml	0.539118177	39.3907	0	0.150386
graph22.gml	0.473098911	39.34826	0	0.361247
graph23.gml	0.513098128	39.18918	0	0.11361
graph24.gml	0.530268619	39.33178	0	0.178551
graph25.gml	0.515310518	39.42718	0	0.048067
graph26.gml	0.511061947	39.51398	0	0.032754
graph27.gml	0.5	39.22671	0	0.204275
graph28.gml	0.498892582	39.33876	0	0.078777
graph29.gml	0.570189521	39.32946	1	0
graph30.gml	0.579665596	39.62732	0	0.077602
graph31.gml	0.513098128	39.01367	0	0.131469
graph32.gml	0.697616053	39.50465	1	0
graph33.gml	0.493010416	37.33667	0	0.173089
graph34.gml	0.5	39.38155	0	0.082357
graph35.gml	0.461508341	39.31291	0	0.17615
graph36.gml	0.530268619	39.20396	0	0.142476
graph37.gml	0.504072363	39.22295	0	0.017928
graph38.gml	0.617472003	39.14212	1	0
graph39.gml	0.5	39.12077	0	0.003006
graph40.gml	0.5	39.06516	0	0.029711

graph41.gml	0.645430339	39.1523	0	0.023432
graph42.gml	0.673408254	39.06701	1	0
graph43.gml	0.530268619	39.05741	0	0.075882
graph44.gml	0.513098128	38.80387	0	0.007198
graph45.gml	0.5	39.21909	0	6.66E-16
graph46.gml	0.705654319	39.23465	1	0
graph47.gml	0.661140996	39.33981	1	0
graph48.gml	0.480547371	39.35599	0	0.071447
graph49.gml	0.5	39.2262	0	0.12159
graph50.gml	0.469653066	39.1254	0	0.060694
graph51.gml	0.544647927	39.37805	1	0
graph52.gml	0.611626547	39.16243	1	0
graph53.gml	0.637568281	39.09725	0	0.018019
graph54.gml	0.530268619	39.13379	0	0.182677
graph55.gml	0.497787611	39.30561	0	0.042229
graph56.gml	0.521731097	39.33061	0	0.103367
graph57.gml	0.5	39.43561	0	0.169973
graph58.gml	0.664558697	39.23395	0	0.119119
graph59.gml	0.566821991	39.38742	0	0.173065
graph60.gml	0.461782442	39.23694	0	0.353613
graph61.gml	0.494694181	39.34047	0	0.209602
graph62.gml	0.709314306	39.13668	1	0
graph63.gml	0.5	39.25733	0	0.150404
graph64.gml	0.484337066	39.29027	0	0.112902
graph65.gml	0.498892582	39.15374	0	0.002215
graph66.gml	0.497435195	39.26986	0	0.113718
graph67.gml	0.513098128	38.9764	0	0.200323
graph68.gml	0.524629963	39.1451	0	0.10504
graph69.gml	0.486549456	39.14723	0	0.186706
graph70.gml	0.5	39.27724	0	0.230514
graph71.gml	0.487683795	39.34684	0	0.311555
graph72.gml	0.474958885	39.15794	0	0.157205
graph73.gml	0.58107526	39.18618	0	0.167815
graph74.gml	0.514173731	39.03171	0	0.250751
graph75.gml	0.577433628	39.08218	0	0.040191
graph76.gml	0.434548516	39.36102	0	0.45051
graph77.gml	0.485844624	38.68215	0	0.134491
graph78.gml	0.546664529	39.32922	1	0
graph79.gml	0.5	39.60411	0	0.081629
graph80.gml	0.474958885	39.04164	0	0.04586
graph81.gml	0.497787611	38.96409	0	0.019322
graph82.gml	0.518481038	39.08616	0	0.161538
graph83.gml	0.48922072	39.24732	0	0.230303
graph84.gml	0.5	39.37891	0	0.112771
graph85.gml	0.514077062	38.96517	0	0.087792

graph86.gml	0.564657324	39.29065	1	0
graph87.gml	0.543003319	39.03219	0	0.071866
graph88.gml	0.5	38.58756	0	0.11778
graph89.gml	0.52805623	39.07905	1	0
graph90.gml	0.513274336	39.1596	0	0.233271
graph91.gml	0.56120169	39.18099	0	0.002126
graph92.gml	0.479383664	39.0393	0	0.260227
graph93.gml	0.498892582	39.39182	0	0.008791
graph94.gml	0.497787611	39.1516	0	0.193286
graph95.gml	0.441518521	38.96851	0	0.181036
graph96.gml	0.566752242	39.06955	1	0
graph97.gml	0.5	38.87737	0	0.119683
graph98.gml	0.535798369	38.93303	0	0.171793
graph99.gml	0.498414128	38.84834	0	0.120687

Tabla 10:Resultados para cada instancia Alpha > Beta

3. Siendo Alpha < Beta

Instancia	Modularidad	Time	Best	Dev
graph0.gml	0.547086694	39.2712	0	0.132367
graph1.gml	0.5	39.51179	0	0.119683
graph2.gml	0.76337223	39.21434	1	0
graph3.gml	0.562397212	39.61396	1	0
graph4.gml	0.485471405	39.3886	0	0.251544
graph5.gml	0.502671264	39.47921	0	0.108031
graph6.gml	0.488761845	39.07071	0	0.266163
graph7.gml	0.561583474	39.05768	1	0
graph8.gml	0.765682512	39.1642	1	0
graph9.gml	0.526548673	39.23194	0	0.167729
graph10.gml	0.595934245	39.19138	1	0
graph11.gml	0.5	39.32604	0	0.170802
graph12.gml	0.564657324	39.40628	0	0.154487
graph13.gml	0.486549456	39.35317	0	0.234139
graph14.gml	0.52805623	39.22175	0	0.08187
graph15.gml	0.584090375	39.35722	0	0.1457
graph16.gml	0.476279221	39.56534	0	0.195421
graph17.gml	0.465228287	39.21061	0	0.149626
graph18.gml	0.631166056	39.4855	1	0
graph19.gml	0.486549456	39.1337	0	0.070504
graph20.gml	0.513098128	39.46409	0	0.024892
graph21.gml	0.6345446	39.27956	1	0
graph22.gml	0.740659752	39.56318	1	0
graph23.gml	0.545579137	38.86464	0	0.057498
graph24.gml	0.645528232	39.5012	1	0
graph25.gml	0.541330566	39.38462	1	0
graph26.gml	0.504424779	39.94566	0	0.045316
graph27.gml	0.628357741	39.37904	1	0
graph28.gml	0.541554497	39.25979	1	0
graph29.gml	0.502036181	39.34071	0	0.119528
graph30.gml	0.531314854	39.21911	0	0.154541
graph31.gml	0.590765477	39.17827	1	0
graph32.gml	0.50406135	39.14927	0	0.277452
graph33.gml	0.506637168	37.18466	0	0.150233
graph34.gml	0.544874305	38.96211	1	0
graph35.gml	0.560184823	38.88703	1	0
graph36.gml	0.614953696	39.0286	0	0.005527
graph37.gml	0.513274336	39.07524	1	0
graph38.gml	0.51859851	40.28933	0	0.160126
graph39.gml	0.501507557	39.39687	1	0
graph40.gml	0.515310518	39.14767	1	0

graph41.gml	0.5	39.25001	0	0.243475
graph42.gml	0.513098128	39.19277	0	0.238058
graph43.gml	0.5	39.25934	0	0.128632
graph44.gml	0.516818075	39.05864	1	0
graph45.gml	0.470886522	39.38453	0	0.058227
graph46.gml	0.514379307	39.20765	0	0.27106
graph47.gml	0.577355314	39.33071	0	0.126729
graph48.gml	0.517522907	39.39284	1	0
graph49.gml	0.569210588	39.3095	1	0
graph50.gml	0.5	39.23186	1	0
graph51.gml	0.529102465	39.18217	0	0.028542
graph52.gml	0.49044561	39.19439	0	0.198129
graph53.gml	0.649267758	39.15393	1	0
graph54.gml	0.516386121	39.24944	0	0.204074
graph55.gml	0.519735296	39.16392	1	0
graph56.gml	0.581877986	39.21579	1	0
graph57.gml	0.602390311	39.3093	1	0
graph58.gml	0.643589944	39.25695	0	0.146913
graph59.gml	0.495575221	39.27325	0	0.277007
graph60.gml	0.714405983	39.0127	1	0
graph61.gml	0.625879816	39.44209	1	0
graph62.gml	0.56120169	39.22947	0	0.208811
graph63.gml	0.588515154	39.38488	1	0
graph64.gml	0.477171274	39.17542	0	0.126027
graph65.gml	0.5	39.14513	1	0
graph66.gml	0.493186624	39.57442	0	0.121287
graph67.gml	0.547086694	39.31471	0	0.147351
graph68.gml	0.586204871	39.49425	1	0
graph69.gml	0.598245751	39.2643	1	0
graph70.gml	0.649784145	39.28698	1	0
graph71.gml	0.708384319	39.0454	1	0
graph72.gml	0.563552353	39.10963	1	0
graph73.gml	0.698252359	39.16374	1	0
graph74.gml	0.68625186	39.02527	1	0
graph75.gml	0.5	39.11498	0	0.168901
graph76.gml	0.790821521	39.24184	1	0
graph77.gml	0.493186624	38.60818	0	0.121412
graph78.gml	0.46019657	38.83657	0	0.158174
graph79.gml	0.544442351	38.75478	1	0
graph80.gml	0.484160858	39.09263	0	0.027375
graph81.gml	0.507595299	39.17147	1	0
graph82.gml	0.533527244	39.13185	0	0.137206
graph83.gml	0.530268619	39.34162	0	0.165722
graph84.gml	0.563552353	39.31261	1	0
graph85.gml	0.470357898	39.04956	0	0.16537

graph86.gml	0.498892582	39.19694	0	0.116468
graph87.gml	0.585048506	39.12659	1	0
graph88.gml	0.566752242	39.1016	1	0
graph89.gml	0.5	39.10966	0	0.053131
graph90.gml	0.5	39.13338	0	0.2531
graph91.gml	0.562397212	39.37101	1	0
graph92.gml	0.648014723	40.01085	1	0
graph93.gml	0.50331736	39.09809	1	0
graph94.gml	0.5	39.1042	0	0.189701
graph95.gml	0.515310518	39.27388	0	0.04416
graph96.gml	0.547086694	39.12348	0	0.034699
graph97.gml	0.492079205	39.41414	0	0.133628
graph98.gml	0.646937896	38.97445	1	0
graph99.gml	0.499647584	39.38317	0	0.118511

Tabla 11:Resultados para cada instancia Alpha < Beta

C. Resultados para cada instancia en la comparación Louvain vs ACO

1. Resultados ACO

Instancia	Modularidad - ACO	Best - ACO	Desv Aco
graph0.gml	0.547086694	1	0
graph1.gml	0.5	0	0.077049
graph2.gml	0.76337223	1	0
graph3.gml	0.562397212	1	0
graph4.gml	0.485471405	0	0.031715
graph5.gml	0.502671264	0	0.005277
graph6.gml	0.488761845	0	0.183316
graph7.gml	0.561583474	1	0
graph8.gml	0.765682512	1	0
graph9.gml	0.526548673	1	0
graph10.gml	0.595934245	1	0
graph11.gml	0.5	0	0.008193
graph12.gml	0.564657324	1	0
graph13.gml	0.486549456	0	0.027788
graph14.gml	0.52805623	1	0
graph15.gml	0.584090375	1	0
graph16.gml	0.476279221	0	0.048892
graph17.gml	0.465228287	0	0.077919
graph18.gml	0.631166056	1	0
graph19.gml	0.486549456	0	0.029071
graph20.gml	0.513098128	0	0.01164
graph21.gml	0.6345446	1	0
graph22.gml	0.740659752	1	0
graph23.gml	0.545579137	1	0
graph24.gml	0.645528232	1	0
graph25.gml	0.541330566	1	0
graph26.gml	0.504424779	1	0
graph27.gml	0.628357741	1	0
graph28.gml	0.541554497	1	0
graph29.gml	0.502036181	1	0
graph30.gml	0.531314854	1	0
graph31.gml	0.590765477	1	0
graph32.gml	0.50406135	0	0.005149
graph33.gml	0.506637168	1	0
graph34.gml	0.544874305	1	0

graph35.gml	0.560184823	1	0
graph36.gml	0.614953696	1	0
graph37.gml	0.513274336	1	0
graph38.gml	0.51859851	1	0
graph39.gml	0.501507557	0	0.002104
graph40.gml	0.515310518	1	0
graph41.gml	0.5	0	0.008741
graph42.gml	0.513098128	0	0.10027
graph43.gml	0.5	0	0.162768
graph44.gml	0.516818075	1	0
graph45.gml	0.470886522	0	0.07153
graph46.gml	0.514379307	1	0
graph47.gml	0.577355314	1	0
graph48.gml	0.517522907	1	0
graph49.gml	0.569210588	1	0
graph50.gml	0.5	0	0.008685
graph51.gml	0.529102465	1	0
graph52.gml	0.49044561	0	0.032459
graph53.gml	0.649267758	1	0
graph54.gml	0.516386121	1	0
graph55.gml	0.519735296	1	0
graph56.gml	0.581877986	1	0
graph57.gml	0.602390311	1	0
graph58.gml	0.643589944	1	0
graph59.gml	0.495575221	0	0.016122
graph60.gml	0.714405983	1	0
graph61.gml	0.625879816	1	0
graph62.gml	0.56120169	1	0
graph63.gml	0.588515154	1	0
graph64.gml	0.477171274	0	0.04636
graph65.gml	0.5	0	0.00124
graph66.gml	0.493186624	0	0.055549
graph67.gml	0.547086694	1	0
graph68.gml	0.586204871	1	0
graph69.gml	0.598245751	1	0
graph70.gml	0.649784145	1	0
graph71.gml	0.708384319	1	0
graph72.gml	0.563552353	1	0
graph73.gml	0.698252359	1	0
graph74.gml	0.68625186	1	0
graph75.gml	0.5	0	0.014118
graph76.gml	0.790821521	1	0
graph77.gml	0.493186624	0	0.01518
graph78.gml	0.46019657	0	0.086115
graph79.gml	0.544442351	1	0

graph80.gml	0.484160858	0	0.035562
graph81.gml	0.507595299	1	0
graph82.gml	0.533527244	1	0
graph83.gml	0.530268619	1	0
graph84.gml	0.563552353	1	0
graph85.gml	0.470357898	0	0.0602
graph86.gml	0.498892582	0	0.005763
graph87.gml	0.585048506	1	0
graph88.gml	0.566752242	1	0
graph89.gml	0.5	0	0.002233
graph90.gml	0.5	0	0.00779
graph91.gml	0.562397212	1	0
graph92.gml	0.648014723	1	0
graph93.gml	0.50331736	0	0.012572
graph94.gml	0.5	0	0.004772
graph95.gml	0.515310518	1	0
graph96.gml	0.547086694	1	0
graph97.gml	0.492079205	0	0.026099
graph98.gml	0.646937896	1	0
graph99.gml	0.499647584	0	0.006692

Tabla 12: Resultados para cada instancia ACO

2. Resultados Louvain

Instancia	Modularidad - Louvain	Mejor - Louvain	Desv (%) - Louvain
graph0.gml	0.547086694	1	0
graph1.gml	0.5	0	0.077049
graph2.gml	0.76337223	1	0
graph3.gml	0.562397212	1	0
graph4.gml	0.485471405	0	0.031715
graph5.gml	0.502671264	0	0.005277
graph6.gml	0.488761845	0	0.183316
graph7.gml	0.561583474	1	0
graph8.gml	0.765682512	1	0
graph9.gml	0.526548673	1	0
graph10.gml	0.595934245	1	0
graph11.gml	0.5	0	0.008193
graph12.gml	0.564657324	1	0
graph13.gml	0.486549456	0	0.027788
graph14.gml	0.52805623	1	0
graph15.gml	0.584090375	1	0
graph16.gml	0.476279221	0	0.048892
graph17.gml	0.465228287	0	0.077919
graph18.gml	0.631166056	1	0
graph19.gml	0.486549456	0	0.029071
graph20.gml	0.513098128	0	0.01164
graph21.gml	0.6345446	1	0
graph22.gml	0.740659752	1	0
graph23.gml	0.545579137	1	0
graph24.gml	0.645528232	1	0
graph25.gml	0.541330566	1	0
graph26.gml	0.504424779	1	0
graph27.gml	0.628357741	1	0
graph28.gml	0.541554497	1	0
graph29.gml	0.502036181	1	0
graph30.gml	0.531314854	1	0
graph31.gml	0.590765477	1	0
graph32.gml	0.50406135	0	0.005149
graph33.gml	0.506637168	1	0
graph34.gml	0.544874305	1	0
graph35.gml	0.560184823	1	0
graph36.gml	0.614953696	1	0
graph37.gml	0.513274336	1	0
graph38.gml	0.51859851	1	0
graph39.gml	0.501507557	0	0.002104
graph40.gml	0.515310518	1	0

graph41.gml	0.5	0	0.008741
graph42.gml	0.513098128	0	0.10027
graph43.gml	0.5	0	0.162768
graph44.gml	0.516818075	1	0
graph45.gml	0.470886522	0	0.07153
graph46.gml	0.514379307	1	0
graph47.gml	0.577355314	1	0
graph48.gml	0.517522907	1	0
graph49.gml	0.569210588	1	0
graph50.gml	0.5	0	0.008685
graph51.gml	0.529102465	1	0
graph52.gml	0.49044561	0	0.032459
graph53.gml	0.649267758	1	0
graph54.gml	0.516386121	1	0
graph55.gml	0.519735296	1	0
graph56.gml	0.581877986	1	0
graph57.gml	0.602390311	1	0
graph58.gml	0.643589944	1	0
graph59.gml	0.495575221	0	0.016122
graph60.gml	0.714405983	1	0
graph61.gml	0.625879816	1	0
graph62.gml	0.56120169	1	0
graph63.gml	0.588515154	1	0
graph64.gml	0.477171274	0	0.04636
graph65.gml	0.5	0	0.00124
graph66.gml	0.493186624	0	0.055549
graph67.gml	0.547086694	1	0
graph68.gml	0.586204871	1	0
graph69.gml	0.598245751	1	0
graph70.gml	0.649784145	1	0
graph71.gml	0.708384319	1	0
graph72.gml	0.563552353	1	0
graph73.gml	0.698252359	1	0
graph74.gml	0.68625186	1	0
graph75.gml	0.5	0	0.014118
graph76.gml	0.790821521	1	0
graph77.gml	0.493186624	0	0.01518
graph78.gml	0.46019657	0	0.086115
graph79.gml	0.544442351	1	0
graph80.gml	0.484160858	0	0.035562
graph81.gml	0.507595299	1	0
graph82.gml	0.533527244	1	0
graph83.gml	0.530268619	1	0
graph84.gml	0.563552353	1	0
graph85.gml	0.470357898	0	0.0602

graph86.gml	0.498892582	0	0.005763
graph87.gml	0.585048506	1	0
graph88.gml	0.566752242	1	0
graph89.gml	0.5	0	0.002233
graph90.gml	0.5	0	0.00779
graph91.gml	0.562397212	1	0
graph92.gml	0.648014723	1	0
graph93.gml	0.50331736	0	0.012572
graph94.gml	0.5	0	0.004772
graph95.gml	0.515310518	1	0
graph96.gml	0.547086694	1	0
graph97.gml	0.492079205	0	0.026099
graph98.gml	0.646937896	1	0
graph99.gml	0.499647584	0	0.006692

Tabla 13: Resultados para cada instancia Louvain

D. Resultados para cada instancia en el ajuste de la Red Neuronal

1. Siendo el learning rate = 0.025

Instancia	NMI	Mejor	Desv (%)	Tiempo (s)
graph0.gml	0.547086694	1	0	44.92506
graph1.gml	0.5	0	0.077049	56.53684
graph2.gml	0.76337223	1	0	32.54532
graph3.gml	0.562397212	1	0	57.27522
graph4.gml	0.485471405	0	0.031715	52.46714
graph5.gml	0.502671264	0	0.005277	41.54886
graph6.gml	0.488761845	0	0.183316	52.74631
graph7.gml	0.561583474	1	0	45.00126
graph8.gml	0.765682512	1	0	58.89203
graph9.gml	0.526548673	1	0	41.27273
graph10.gml	0.595934245	1	0	53.98918
graph11.gml	0.5	0	0.008193	47.78886
graph12.gml	0.564657324	1	0	32.50313
graph13.gml	0.486549456	0	0.027788	40.20996
graph14.gml	0.52805623	1	0	49.84058
graph15.gml	0.584090375	1	0	39.27797
graph16.gml	0.476279221	0	0.048892	54.25048
graph17.gml	0.465228287	0	0.077919	33.34674
graph18.gml	0.631166056	1	0	57.82835
graph19.gml	0.486549456	0	0.029071	40.9607
graph20.gml	0.513098128	0	0.01164	46.82367
graph21.gml	0.6345446	1	0	32.93974
graph22.gml	0.740659752	1	0	43.01262
graph23.gml	0.545579137	1	0	34.99353
graph24.gml	0.645528232	1	0	47.76188
graph25.gml	0.541330566	1	0	34.50496
graph26.gml	0.504424779	1	0	33.72933
graph27.gml	0.628357741	1	0	54.30041
graph28.gml	0.541554497	1	0	42.32371
graph29.gml	0.502036181	1	0	31.72783
graph30.gml	0.531314854	1	0	46.37669
graph31.gml	0.590765477	1	0	39.59155
graph32.gml	0.50406135	0	0.005149	46.19997
graph33.gml	0.506637168	1	0	40.84082
graph34.gml	0.544874305	1	0	57.34154

graph35.gml	0.560184823	1	0	38.03268
graph36.gml	0.614953696	1	0	57.10196
graph37.gml	0.513274336	1	0	42.69611
graph38.gml	0.51859851	1	0	60.11585
graph39.gml	0.501507557	0	0.002104	34.16264
graph40.gml	0.515310518	1	0	41.15861
graph41.gml	0.5	0	0.008741	35.25378
graph42.gml	0.513098128	0	0.10027	46.7947
graph43.gml	0.5	0	0.162768	50.36861
graph44.gml	0.516818075	1	0	35.84461
graph45.gml	0.470886522	0	0.07153	38.06619
graph46.gml	0.514379307	1	0	37.85677
graph47.gml	0.577355314	1	0	36.6577
graph48.gml	0.517522907	1	0	49.09871
graph49.gml	0.569210588	1	0	39.05755
graph50.gml	0.5	0	0.008685	62.31556
graph51.gml	0.529102465	1	0	39.10243
graph52.gml	0.49044561	0	0.032459	51.94831
graph53.gml	0.649267758	1	0	47.613
graph54.gml	0.516386121	1	0	51.86037
graph55.gml	0.519735296	1	0	59.19822
graph56.gml	0.581877986	1	0	34.08382
graph57.gml	0.602390311	1	0	41.79257
graph58.gml	0.643589944	1	0	51.59676
graph59.gml	0.495575221	0	0.016122	38.63842
graph60.gml	0.714405983	1	0	34.86954
graph61.gml	0.625879816	1	0	42.09668
graph62.gml	0.56120169	1	0	53.76421
graph63.gml	0.588515154	1	0	33.0104
graph64.gml	0.477171274	0	0.04636	50.28791
graph65.gml	0.5	0	0.00124	35.31011
graph66.gml	0.493186624	0	0.055549	37.39949
graph67.gml	0.547086694	1	0	52.54458
graph68.gml	0.586204871	1	0	41.20433
graph69.gml	0.598245751	1	0	39.70186
graph70.gml	0.649784145	1	0	33.74442
graph71.gml	0.708384319	1	0	39.48229
graph72.gml	0.563552353	1	0	60.01826
graph73.gml	0.698252359	1	0	61.19813
graph74.gml	0.68625186	1	0	35.41182
graph75.gml	0.5	0	0.014118	33.69124
graph76.gml	0.790821521	1	0	49.08974
graph77.gml	0.493186624	0	0.01518	38.92911
graph78.gml	0.46019657	0	0.086115	34.72451
graph79.gml	0.544442351	1	0	31.74061

graph80.gml	0.484160858	0	0.035562	33.66449
graph81.gml	0.507595299	1	0	38.40906
graph82.gml	0.533527244	1	0	34.17538
graph83.gml	0.530268619	1	0	40.52398
graph84.gml	0.563552353	1	0	55.29378
graph85.gml	0.470357898	0	0.0602	34.78141
graph86.gml	0.498892582	0	0.005763	47.70701
graph87.gml	0.585048506	1	0	38.34925
graph88.gml	0.566752242	1	0	56.29721
graph89.gml	0.5	0	0.002233	42.69727
graph90.gml	0.5	0	0.00779	41.25728
graph91.gml	0.562397212	1	0	41.35238
graph92.gml	0.648014723	1	0	56.72427
graph93.gml	0.50331736	0	0.012572	49.96584
graph94.gml	0.5	0	0.004772	49.23423
graph95.gml	0.515310518	1	0	64.75923
graph96.gml	0.547086694	1	0	42.34099
graph97.gml	0.492079205	0	0.026099	75.5049
graph98.gml	0.646937896	1	0	69.79477
graph99.gml	0.499647584	0	0.006692	37.60197

Tabla 14: Resultados para cada instancia con learning rate = 0.025

2. Siendo learning rate = 0.050

Instancia	NMI	Mejor	Desv (%)	Tiempo (s)
graph0.gml	0.022947	0	0	44.60514
graph1.gml	0.01	0	0.785099	56.66271
graph2.gml	0.643471	0	0	33.27625
graph3.gml	0.032682	0	0	57.77115
graph4.gml	0.01	0	0.95208	49.73034
graph5.gml	0.007755	0	0.276315	41.54205
graph6.gml	0.01	0	0	52.25613
graph7.gml	0.037799	0	0.948841	44.2303
graph8.gml	0.035148	0	0.949667	57.24187
graph9.gml	0.002149	0	0.995518	41.44582
graph10.gml	0.01	0	0.957214	53.42202
graph11.gml	0.01	0	0	48.04155
graph12.gml	0.48442	0	0	33.02889
graph13.gml	0.071935	0	0.890637	40.09644
graph14.gml	0.05278	0	0	48.92161
graph15.gml	0.042445	0	0.61765	39.51552
graph16.gml	0.01	0	0.958838	53.13993
graph17.gml	0.649222	0	0	33.07025
graph18.gml	0.04014	0	0.9408	57.12091
graph19.gml	0.01	0	0.64052	41.54954
graph20.gml	0.043721	0	0.938034	46.92341
graph21.gml	0.506556	0	0	32.59322
graph22.gml	0.036679	0	0.951748	42.09715
graph23.gml	0.01	0	0.98318	35.32825
graph24.gml	0.043635	0	0.932184	47.91357
graph25.gml	0.04749	0	0.274779	34.55622
graph26.gml	0.006009	0	0.875125	33.34422
graph27.gml	0.01	0	0	53.78328
graph28.gml	0.066244	0	0.899786	41.75186
graph29.gml	0.042222	0	0.90731	31.60128
graph30.gml	0.01	1	0	45.70958
graph31.gml	0.01	0	0	39.31126
graph32.gml	0.01	0	0	45.37042
graph33.gml	0.076192	0	0.883416	39.73591
graph34.gml	0.037565	0	0	57.17378
graph35.gml	0.00714	0	0.989272	36.88876
graph36.gml	0.702426	1	0	57.58522
graph37.gml	0.063016	0	0.905151	42.68942
graph38.gml	0.036368	0	0	60.25703
graph39.gml	0.01	0	0	34.31909
graph40.gml	0.028254	1	0	40.69636

graph41.gml	0.061298	0	0.909622	35.37636
graph42.gml	0.039418	0	0	46.52939
graph43.gml	0.01	0	0	50.1758
graph44.gml	0.094988	0	0	36.05453
graph45.gml	0.004206	0	0.992844	38.47797
graph46.gml	0.021757	1	0	38.20808
graph47.gml	0.01	0	0.650197	36.47889
graph48.gml	0.037135	0	0.948699	48.66928
graph49.gml	0.013343	0	0.487471	39.07147
graph50.gml	0.033357	0	0.952352	61.56899
graph51.gml	0.01	0	0.968074	38.27745
graph52.gml	0.01	0	0	52.13593
graph53.gml	0.01	0	0	48.71365
graph54.gml	0.034614	1	0	52.84186
graph55.gml	0.011312	0	0	58.24361
graph56.gml	0.071952	0	0.900102	34.21161
graph57.gml	0.003561	0	0.841986	41.73215
graph58.gml	0.043981	0	0.936071	50.6135
graph59.gml	0.003276	0	0.764408	37.53922
graph60.gml	0.419938	0	0.071974	35.32872
graph61.gml	0.01	0	0.078985	42.34427
graph62.gml	0.039382	0	0	54.12751
graph63.gml	0.649708	0	0	33.27905
graph64.gml	0.01	0	0	50.20809
graph65.gml	0.01	0	0.985035	34.71682
graph66.gml	0.13852	0	0.237971	37.51229
graph67.gml	0.038987	0	0	51.95995
graph68.gml	0.026812	1	0	41.31745
graph69.gml	0.672099	0	0	40.19966
graph70.gml	0.065348	0	0.859744	34.36275
graph71.gml	0.025385	1	0	39.45983
graph72.gml	0.041741	0	0.937388	58.87214
graph73.gml	0.011446	0	0.872203	60.36934
graph74.gml	0.01	0	0.972748	35.5871
graph75.gml	0.512343	0	0.263928	33.6374
graph76.gml	0.01	0	0	48.73914
graph77.gml	0.02189	0	0	38.7423
graph78.gml	0.01	0	0	34.57804
graph79.gml	0.751759	0	0	31.7226
graph80.gml	0.01	0	0	33.56281
graph81.gml	0.686514	0	0	39.03446
graph82.gml	0.513157	0	0	33.66217
graph83.gml	0.013272	0	0	40.77456
graph84.gml	0.01	0	0	54.70937
graph85.gml	0.06917	0	2.01E-16	35.10205

graph86.gml	0.030243	0	0	47.54957
graph87.gml	0.016361	0	0.515873	38.54749
graph88.gml	0.01	0	0.956487	55.31188
graph89.gml	0.091563	0	0.862347	42.79051
graph90.gml	0.024364	0	0.529609	42.99829
graph91.gml	0.004495	0	0.793911	41.44939
graph92.gml	0.01	0	0	57.75748
graph93.gml	0.647989	1	0	46.19242
graph94.gml	0.136228	1	0	49.06726
graph95.gml	0.01	0	0	62.91801
graph96.gml	0.391041	0	0.381012	40.53575
graph97.gml	0.0333	0	0	76.21342
graph98.gml	0.031952	0	0	70.77052
graph99.gml	0.027965	1	0	37.27887

Tabla 15: Resultados para cada instancia con learning rate = 0.050

3. Siendo learning rate = 0.075

Instancia	NMI	Mejor	Desv (%)	Tiempo (s)
graph0.gml	0.01	0	0.564219	44.62984
graph1.gml	0.046533	0	0	56.75325
graph2.gml	0.643471	0	0	32.95111
graph3.gml	0.032682	0	0	57.61295
graph4.gml	0.01	0	0.95208	49.6472
graph5.gml	0.010716	1	0	41.78407
graph6.gml	0.01	0	0	52.1219
graph7.gml	0.037799	0	0.948841	44.40779
graph8.gml	0.017962	0	0.974278	56.93959
graph9.gml	0.01	0	0.979145	41.20075
graph10.gml	0.01	0	0.957214	52.94542
graph11.gml	0.01	0	0	48.71622
graph12.gml	0.031977	0	0.933988	33.06864
graph13.gml	0.071935	0	0.890637	40.32905
graph14.gml	0.034863	0	0.339465	48.49006
graph15.gml	0.022558	0	0.796794	39.00525
graph16.gml	0.01	0	0.958838	52.87642
graph17.gml	0.168411	0	0.740595	32.94308
graph18.gml	0.04014	0	0.9408	57.81661
graph19.gml	0.012291	0	0.558153	41.12899
graph20.gml	0.043721	0	0.938034	47.31938
graph21.gml	0.01	0	0.980259	32.22576
graph22.gml	0.01	0	0.986845	42.32749
graph23.gml	0.594518	1	0	34.96761
graph24.gml	0.022378	0	0.965221	47.77257
graph25.gml	0.065483	1	0	34.39666
graph26.gml	0.048122	1	0	33.56201
graph27.gml	0.01	0	0	54.0653
graph28.gml	0.066244	0	0.899786	42.14191
graph29.gml	0.080039	0	0.824287	31.92273
graph30.gml	0.01	0	0	46.15168
graph31.gml	0.01	0	0	39.38607
graph32.gml	0.01	0	0	45.01623
graph33.gml	0.076192	0	0.883416	39.79334
graph34.gml	0.037565	0	0	57.43745
graph35.gml	0.035466	0	0.946714	36.93354
graph36.gml	0.035241	0	0.94983	56.75497
graph37.gml	0.063016	0	0.905151	42.5828
graph38.gml	0.036368	0	0	60.99154
graph39.gml	0.01	0	0	34.01895
graph40.gml	0.028254	0	0	40.94929

graph41.gml	0.04578	0	0.932502	35.13089
graph42.gml	0.039418	0	0	46.04333
graph43.gml	0.01	0	0	49.75423
graph44.gml	0.094988	0	0	36.43121
graph45.gml	0.028841	0	0.950927	38.12878
graph46.gml	0.01	0	0.540387	37.55113
graph47.gml	0.01	0	0.650197	36.88866
graph48.gml	0.037135	0	0.948699	49.44338
graph49.gml	0.021706	0	0.166244	39.37219
graph50.gml	0.033357	0	0.952352	61.72277
graph51.gml	0.01	0	0.968074	38.48921
graph52.gml	0.01	0	0	51.81072
graph53.gml	0.01	0	0	48.23216
graph54.gml	0.017721	0	0.488033	52.37299
graph55.gml	0.011312	0	0	58.3014
graph56.gml	0.057336	0	0.920396	34.17603
graph57.gml	0.022536	1	0	41.50263
graph58.gml	0.022528	0	0.967254	50.69199
graph59.gml	0.013905	1	0	38.22478
graph60.gml	0.452507	1	0	35.06325
graph61.gml	0.010858	1	0	42.28793
graph62.gml	0.039382	0	0	54.29739
graph63.gml	0.182009	0	0.71986	33.40497
graph64.gml	0.01	0	0	50.40474
graph65.gml	0.668237	1	0	36.21174
graph66.gml	0.045019	0	0.752338	37.09236
graph67.gml	0.038987	0	0	52.03263
graph68.gml	0.026812	0	0	41.80548
graph69.gml	0.070737	0	0.894752	39.59776
graph70.gml	0.046223	0	0.900792	33.8932
graph71.gml	0.025385	0	0	39.31474
graph72.gml	0.041741	0	0.937388	58.7191
graph73.gml	0.005858	0	0.934598	61.20124
graph74.gml	0.025273	0	0.931126	35.5449
graph75.gml	0.030337	0	0.956415	33.48155
graph76.gml	0.01	0	0	48.44892
graph77.gml	0.02189	0	0	39.11369
graph78.gml	0.01	0	0	34.89634
graph79.gml	0.751759	0	0	31.6081
graph80.gml	0.01	0	0	33.74454
graph81.gml	0.069495	0	0.898771	38.09235
graph82.gml	0.479804	0	0.064996	33.75423
graph83.gml	0.013272	0	0	40.31746
graph84.gml	0.01	0	0	54.57556
graph85.gml	0.06917	0	2.01E-16	34.56572

graph86.gml	0.030243	0	0	47.91279
graph87.gml	0.016361	0	0.515873	38.69896
graph88.gml	0.01	0	0.956487	63.85545
graph89.gml	0.091563	0	0.862347	42.45096
graph90.gml	0.018054	0	0.651442	41.75425
graph91.gml	0.021812	1	0	43.1859
graph92.gml	0.01	0	0	57.06084
graph93.gml	0.069189	0	0.893225	49.03987
graph94.gml	0.019149	0	0.859437	47.77028
graph95.gml	0.01	0	0	65.24293
graph96.gml	0.391041	0	0.381012	41.96682
graph97.gml	0.0333	0	0	75.59681
graph98.gml	0.016328	0	0.488982	70.53373
graph99.gml	0.027965	0	0	37.58821

Tabla 16: Resultado para cada instancia con learning rate = 0.075

4. Siendo learning rate = 0.100

Instancia	NMI	Mejor	Desv (%)	Tiempo (s)
graph0.gml	0.022947	0	0	44.8579
graph1.gml	0.023781	0	0.488939	55.95246
graph2.gml	0.180986	0	0.718735	32.67051
graph3.gml	0.0167	0	0.489026	57.71741
graph4.gml	0.01	0	0.95208	49.71193
graph5.gml	0.007755	0	0.276315	41.69861
graph6.gml	0.01	0	0	51.99712
graph7.gml	0.01	0	0.986466	44.0888
graph8.gml	0.035148	0	0.949667	57.35181
graph9.gml	0.01	0	0.979145	41.47748
graph10.gml	0.01	0	0.957214	53.54518
graph11.gml	0.01	0	0	48.03314
graph12.gml	0.033525	0	0.930794	33.37115
graph13.gml	0.071935	0	0.890637	40.37551
graph14.gml	0.034863	0	0.339465	48.84584
graph15.gml	0.022558	0	0.796794	39.11014
graph16.gml	0.01	0	0.958838	53.6598
graph17.gml	0.168411	0	0.740595	33.09799
graph18.gml	0.04014	0	0.9408	57.27669
graph19.gml	0.027818	0	0	41.53878
graph20.gml	0.043721	0	0.938034	47.40786
graph21.gml	0.506556	0	0	32.43246
graph22.gml	0.01	0	0.986845	42.40885
graph23.gml	0.01	0	0.98318	35.11911
graph24.gml	0.643426	1	0	48.3814
graph25.gml	0.04749	0	0.274779	34.37555
graph26.gml	0.048122	0	0	33.74393
graph27.gml	0.01	0	0	53.99873
graph28.gml	0.066244	0	0.899786	42.03215
graph29.gml	0.080039	0	0.824287	31.88065
graph30.gml	0.01	0	0	45.37093
graph31.gml	0.01	0	0	39.77783
graph32.gml	0.01	0	0	45.08931
graph33.gml	0.076192	0	0.883416	39.72741
graph34.gml	0.037565	0	0	57.90199
graph35.gml	0.035466	0	0.946714	36.83558
graph36.gml	0.018017	0	0.974351	56.82233
graph37.gml	0.063016	0	0.905151	43.01189
graph38.gml	0.018578	0	0.489186	59.53365
graph39.gml	0.01	0	0	34.0816
graph40.gml	0.028254	0	0	40.85909

graph41.gml	0.023269	0	0.965691	35.03719
graph42.gml	0.020236	0	0.486616	46.39667
graph43.gml	0.01	0	0	49.89687
graph44.gml	0.049051	0	0.483611	36.29143
graph45.gml	0.587715	1	0	38.06758
graph46.gml	0.011295	0	0.480857	38.18408
graph47.gml	0.028587	1	0	36.77713
graph48.gml	0.01904	0	0.973696	48.68566
graph49.gml	0.026034	1	0	39.37488
graph50.gml	0.017037	0	0.975664	61.71578
graph51.gml	0.313224	1	0	39.42987
graph52.gml	0.01	0	0	51.11609
graph53.gml	0.01	0	0	48.34185
graph54.gml	0.034614	0	0	52.94819
graph55.gml	0.01	0	0.115988	57.65953
graph56.gml	0.057336	0	0.920396	34.36517
graph57.gml	0.01153	0	0.488385	41.03882
graph58.gml	0.022528	0	0.967254	50.52283
graph59.gml	0.013905	0	0	38.21634
graph60.gml	0.048118	0	0.893663	35.00115
graph61.gml	0.01	0	0.078985	42.16266
graph62.gml	0.039382	0	0	53.73474
graph63.gml	0.182009	0	0.71986	32.88765
graph64.gml	0.01	0	0	50.72235
graph65.gml	0.01	0	0.985035	35.04109
graph66.gml	0.024059	0	0.867646	37.21324
graph67.gml	0.019961	0	0.488017	52.39884
graph68.gml	0.026812	0	0	41.89916
graph69.gml	0.070737	0	0.894752	39.57778
graph70.gml	0.446441	0	0.041811	34.00401
graph71.gml	0.01	0	0.606073	39.32009
graph72.gml	0.021328	0	0.968008	58.8618
graph73.gml	0.089565	1	0	61.00693
graph74.gml	0.039207	0	0.893154	35.82325
graph75.gml	0.059311	0	0.914789	33.44422
graph76.gml	0.01	0	0	48.4068
graph77.gml	0.01134	0	0.481959	38.83319
graph78.gml	0.01	0	0	34.59865
graph79.gml	0.751759	0	0	31.34642
graph80.gml	0.01	0	0	33.92621
graph81.gml	0.035834	0	0.947803	38.07143
graph82.gml	0.485026	0	0.054819	33.78019
graph83.gml	0.010882	0	0.180129	40.32651
graph84.gml	0.01	0	0	54.58066
graph85.gml	0.043604	0	0.369619	34.80149

graph86.gml	0.015886	0	0.474719	47.97097
graph87.gml	0.033796	1	0	38.59026
graph88.gml	0.01	0	0.956487	70.28928
graph89.gml	0.091563	0	0.862347	41.46557
graph90.gml	0.051796	1	0	42.14974
graph91.gml	0.021812	0	0	43.10707
graph92.gml	0.01	0	0	56.88868
graph93.gml	0.069189	0	0.893225	49.7653
graph94.gml	0.019149	0	0.859437	49.45486
graph95.gml	0.01	0	0	63.64128
graph96.gml	0.024861	0	0.960647	40.57893
graph97.gml	0.017004	0	0.489379	75.43537
graph98.gml	0.031952	0	0	68.3654
graph99.gml	0.01434	0	0.487239	37.17148

Tabla 17: Resultados para cada instancia con learning rate = 0.100

E. Acceso al repositorio con el código del proyecto

Url: https://github.com/Agudroid/CommunityDetection