Facultad
de
Ciencias

Algoritmos de optimización por colonia de hormigas para la maximización de la influencia en grafos

(Ant colony optimization algorithms for influence maximization in graphs)

Trabajo de Fin de Grado
para acceder al

Grado en Ingeniería Informática

Autor: Víctor López Blanco

Director: Camilo Palazuelos Calderon

Julio - 2023

Índice

Índice de algoritmos

Índice de figuras

Índice de tablas

Resumen

En los tiempos que vivimos, nos rodea una gran afluencia de información que podemos usar para nuestros propósitos. Es posible modelar el entorno que nos rodea en grafos para analizar las distintas relaciones y sistemas que se hallan presentes a nuestro alrededor. Dentro de la gran cantidad de problemas que pueden abarcar los grafos hay un conjunto que son recogidos por el problema de la maximización de la influencia. El problema tiene aplicaciones en marketing viral, por ejemplo. Dadas una red social y una estimación de en qué medida quién influye a quién, para una empresa que pretende comercializar un nuevo producto es interesante saber qué miembros influyentes de la red social pueden provocar una mayor aceptación del producto al darle visibilidad. El inconveniente de este problema es que es NP-duro, y por lo tanto encontrar soluciones para problemas con grandes cantidades de datos puede ser computacionalmente muy costoso. No obstante, existen formas de paliar esta desventaja, con el uso de heurísticas, que a cambio de darnos soluciones aproximadas, tendrán una complejidad temporal mucho menor. En este caso nos centraremos en la metaheurística de colonia de hormigas y experimentaremos con distintas variantes de este algoritmo, con el objetivo de diseñar e implementar un nuevo algoritmo de optimización que supere, en tiempo de ejecución o soluciones encontradas, a la versión ACO-IM (Singh et al. 2020) de la que se partirá.

Palabras clave

Grafos; Metaheurística; Maximización de la influencia; Modelos de difusión; Colonia de hormigas

Abstract

Nowadays, we are surrounded by a large amount of information, which we can use for our purposes. It's possible to model our surroundings in influence graphs, to analyze various relationships and systems that exist around us. There are a lot of problems that we can include in the scope of influence graphs, but we are going to focus on those related to the problem of maximizing influence. This problem has applications in viral marketing, for example. Given a social network and an estimate of who influences whom, it is interesting to know for a company which influential members in the social network can trigger a major acceptance of the product by giving it visibility. A big drawback of this problem is that it is NP-hard, which means that finding solutions for problems with a large collection of data can be computationally complex. Nonetheless, we can circumvent this by using heuristics, which will give approximated solutions in a much better time. In this case, we will focus on the metaheuristic of Ant Colony, we will experiment with various variants of this algorithm, with the objective of designing and implementing a new optimization algorithm that exceeds, in execution time or solutions found, the ACO-IM (Singh et al. 2020) version from which we will start.

Keywords

Graphs; Metaheuristic; Influence maximization; Diffusion model; Ant colony

1. Introducción

1.1. Motivación

Es imposible no notar la gran influencia que el *big data* tiene sobre nuestras vidas, y la gran cantidad de información que nos rodea, por lo que es natural que distintos enfoques de la informática se centren en el análisis de este conjunto de datos. Una de las vías de estudio es comprender y predecir como se expande y difunde la información en nuestro entorno, y ser capaces de encontrar la forma de conseguir aumentar o disminuir este flujo de información. En eso se centra el problema de la maximización de la influencia: en ser capaces de encontrar aquellos nodos que consigan una mayor propagación de la influencia a lo largo del grafo.

La maximización de la influencia se trata de un problema NP-duro. Dichos problemas NP-duros (suponiendo que $P \neq NP$) pueden ser reducidos polinomialmente en problemas pertenecientes a NP, donde los problemas pueden ser resueltos en tiempo polinómico por algoritmos indeterministas. Por ello, uno de los enfoques realizados a la hora de tratar con este problema es el uso de metodologías heurísticas, que a pesar de conseguir soluciones no del todo óptimas sin garantías de aproximación, consiguen mejores tiempos de ejecución y mayor escalabilidad.

Existen numerosas formas de tratar el problema de la maximización de la influencia (Banerjee et al. 2020), pero para este estudio nos centraremos en las metaheurísticas, las cuales suelen partir de ideas más abstractas como el comportamiento de ciertos elementos de la naturaleza, siendo un ejemplo la genética. En particular usaremos la familia de metaheurísticas de colonia de hormigas, donde partiremos del algoritmo ACO-IM (Singh et al. 2020), luego lo reduciremos a su versión más básica, para tener una base donde implementar mejoras, y por último probaremos distintas medidas de actualizar las feromonas (Pérez-Carabaza et al. 2022) con el fin de ver una cierta mejora en los resultados o tiempo.

1.2. Hipótesis del trabajo

La hipótesis de este Trabajo de Fin de Grado es que existe la posibilidad de mejorar el algoritmo ACO-IM (Singh et al. 2020) a través de utilizar diferentes variantes en la parte de actualización de feromonas con el objetivo de mejorar en tiempo de ejecución o de calidad de las soluciones el problema de la maximización de la influencia estudiado.

1.3. Objetivos

El principal objetivo de este Trabajo de Fin de Grado es demostrar la hipótesis propuesta anteriormente, lo que se puede desglosar en varios objetivos específicos que se pretenden perseguir:

- Implementar una versión del algoritmo ACO-IM (Singh et al. 2020) en Python, así como replicar la serie de resultados conseguidos por los autores.
- Proponer e implementar cambios sobre el algoritmo ACO-IM, en primer lugar reduciéndolo a una versión básica donde se prescinda de diferentes mejoras y posteriormente explorando distintas variantes para llevar a cabo la tarea de actualizar feromonas.

- Evaluar y comparar cada algoritmo analizado por cada modelo de difusión visto previamente, teniendo en cuenta su rendimiento y el tiempo consumido.

1.4. Estructura del documento

En el capítulo 2 ahondaremos en la terminología que se usará a lo largo del trabajo, haciendo clara una notación para referirnos a los conceptos que vayamos introduciendo sobre teoría de grafos, maximización de la influencia y modelos de difusión.

Posteriormente en el capítulo 3 profundizaremos en la metaheurística de colonia de hormigas, estudiando su forma más básica, para después analizar los cambios se que proponen en el algoritmo ACO-IM (Singh et al. 2020).

Continuaremos analizando en el capítulo 4 la propuesta de mejora que propondremos, viendo mecanismos de actualización de feromonas alternativos y qué estrategias y decisiones de diseño se han tomado a la hora de implementar los algoritmos, ya sea por temas de optimización de rendimiento o de eficiencia.

Seguiremos en el capítulo 5 estudiando cómo se han realizado las distintas pruebas sobre los grafos seleccionados, se mostrarán los resultados a los que se han llegado y qué podemos extraer de ellos.

Por último, en el capítulo 6 finalizaremos con las conclusiones que se pueden sacar de la realización de este trabajo, así como qué vías se pueden tomar a la hora de retomarlo en trabajos futuros.

2. Conceptos básicos y notación

Con la finalidad de poder describir con detalle el problema de la maximización de la influencia y la metaheurística de colonia de hormigas, se describirán los conocimientos base esenciales en teoría de grafos.

2.1. Teoría de grafos

Un grafo $G = (V, E)$ es un conjunto de nodos $V(G) \neq \varnothing$ unidos por un conjunto de aristas $E \subseteq \{(u, v) \in V(G) \times V(G) \mid u \neq v\}$. Para este estudio, trataremos con grafos donde cada nodo representará a un usuario y cada arista representará una relación. Cada nodo puede estar unido a uno o más nodos por aristas. Para representar los grafos, usaremos grafos dirigidos; si el nodo u está unido al nodo v por la arista (u, v), v no tiene por qué estar unido a u, a no ser que también exista la arista (v, u).

A su vez, necesitaremos un conjunto de pesos $W(G) : E(G) \to [0, 1]$ asignados a cada una de las aristas. Los pesos representarán la probabilidad de difusión, siendo ésta la facilidad de propagación de la influencia entre los dos nodos unidos por la arista. Cuanto mayor es este valor en la arista (u, v), mayor influencia ejercerá el nodo u sobre el nodo v y será mucho más probable que el nodo v acabe influenciado.

A su vez, nos es necesario conocer qué relaciones hay entre los nodos con el fin de estudiar las distintas interacciones entre los usuarios. Para ello, dentro del contexto del grafo G, podemos definir la vecindad $\mathcal{N}(u)$ del nodo u cómo el conjunto de nodos que

están unidos al nodo u. Para grafos dirigidos, hablaremos de vecindad de salida del nodo $\mathcal{N}^{out}(u)$ como el conjunto de nodos a los que el nodo u está conectado por aristas salientes.

$$\mathcal{N}^{out}(u) = \{v \mid (u, v) \in E(G)\} \tag{1}$$

Asimismo, también existirá la vecindad de entrada del nodo $\mathcal{N}^{in}(u)$, definido como el conjunto de nodos a los que el nodo u está conectado por aristas entrantes.

$$\mathcal{N}^{in}(u) = \{v \mid (v, u) \in E(G)\} \tag{2}$$

Una vez introducido el concepto de vecindad, es natural hablar a su vez del grado de un nodo para cuantificar dicha vecindad. En un grafo G, el grado de un nodo $D(u)$ está definido por el tamaño de su vecindad $D(u) = |\mathcal{N}(u)|$. En grafos dirigidos, podremos hablar de grado de salida $D^{out}(u)$ de un nodo u como el tamaño de su vecindad de salida $D^{out}(u) = |\mathcal{N}^{out}(u)|$, y de su grado de entrada $D^{in}(u)$ como el modulo de su vecindad de entrada. $D^{in}(u) = |\mathcal{N}^{in}(u)|$.

2.2. Maximización de la influencia

Describiremos el problema que nos ocupa, así como algún concepto que usaremos a lo largo del trabajo para referirnos a la terminología del problema, y posteriormente nos adentraremos en el estado del arte.

Planteamiento del problema

La maximización de la influencia en grafos es un problema computacional que, dados un grafo $G = (V, E)$, un número entero positivo k y un modelo de difusión D, consiste en encontrar uno de los subconjuntos de V de k vértices que maximicen el número de nodos alcanzados (o influidos o infectados, la terminología depende del contexto) tras un proceso estocástico de difusión de acuerdo con D.

Denominaremos a los k nodos que elegiremos para iniciar la difusión de influencia nodos semilla $\mathcal{S}$. Todo nodo que sea influido, ya sea por formar parte de los nodos semilla $\mathcal{S}$ o por ser posteriormente influidos les denominaremos nodos activos $\mathcal{A}$. Dentro de los modelos de difusión D que estudiaremos, un nodo activo $u \in \mathcal{A}$ no puede pasar a estado inactivo. Por otra parte, a la cantidad de nodos a los que habremos influido al final del proceso de difusión la denominaremos propagación de la influencia $\sigma(\mathcal{S})$.

Estado del arte

Una vez entendido el problema de la maximización de la influencia, podemos pasar a explicar de donde vienen todos estos conceptos. Debemos remontarnos al año 2001 donde Domingos y Richardson 2001 forjaron este problema desde un enfoque de optimización, donde lo resuelven modelizándolo como campos aleatorio de Markov, completamente dirigido al mundo del marketing. Es necesario esperar hasta 2003 para que Kempe et al. 2003 den un enfoque computacional a todo este problema. A su vez, probaron que el problema es NP-duro (Banerjee et al. 2020), tanto si lo tratamos con los modelos de difusión IC o WC, donde pasará a ser una instancia del *Set Covering Problem SCP* (buscar el subconjunto de S más pequeño estén todos los elementos del conjunto de elementos llamado universo, siendo S una colección de conjuntos formados por elementos del universo), o si lo estudiamos

bajo el modelo de difusión LT, que pasaría a ser una instancia del *Vertex Cover Problem* (donde debe hallarse el menor número de nodos para alcanzar la completa cobertura del grafo, es decir, que todas las aristas del grafo este unidas a los nodos seleccionados).

Desde que fue propuesto como problema computacional (Kempe et al. 2003) se ha abordado la tarea desde muchos focos distintos, probando todo tipo de métodos con la finalidad de encontrar un algoritmo capaz de encontrar soluciones de la forma más eficiente posible. Podemos distinguir distintas tendencias o variantes (Banerjee et al. 2020):

Para empezar estarían los **algoritmos de aproximación con garantías de comportamiento**, que como indica el nombre son aquellos que dan una solución no del todo óptima, pero siempre dando el error acotado ϵ que puede llegar a tener. El resto de tipo de algoritmos no darán este error, y por lo tanto, no tendremos la seguridad de que su rendimiento sea el esperado. Aquí se encontraría por ejemplo el primer algoritmo propuesto (Kempe et al. 2003), tratándose de un algoritmo voraz, que trata de coger los mejores nodos posible que, al añadirlos a la solución, le den una mayor propagación de la influencia $\sigma(\mathcal{S})$ posible. Para ello tiene que ir calculando cada vez la propagación de la influencia $\sigma(\mathcal{S})$ mediante simulaciones de Monte Carlo, por lo que es muy costoso computacionalmente. Da como aproximación un error de $(1 - \frac{1}{e} - \epsilon)$ con $\epsilon > 0$. Este error acotado lo que da es un ratio que indica como de mala será la solución aproximada respecto a la óptima.

Por otra parte estarían los **algoritmos heurísticos**, que tratan de circundar el inconveniente de tener un problema NP-duro y dan soluciones más eficientes en el sentido temporal, aunque sus soluciones no son tan eficientes. Puede ser algo tan simple como el algoritmo random, que se dedica a seleccionar nodos al azar (usualmente usado para compararlo con otros algoritmos y ver el rendimiento). Un ejemplo de uno menos trivial sería el Degree Discount (Chen et al. 2009), donde vamos escogiendo los nodos con mayor grado $D(u)$, descontando del grado todo aquel nodo que ya haya sido escogido.

A estos le seguirían los **algoritmos metaheurísticos**, que siguen el mismo concepto que los heurísticos, pero como decíamos en la introducción aplicando ideas abstractas basados en comportamientos vistos en la naturaleza. Nosotros nos centraremos en uno de ellos, siendo toda la familia de metaheurísticas de colonia de hormigas, que veremos más a fondo en su sección, pero también existen otros aplicados al problema de influencia, por ejemplo de carácter genético.

Por último existen los **algoritmos basados en las comunidades**, siendo las comunidades subgrafos con nodos mayoritariamente conectados entre ellos, con mucha mayor densidad de conectividad entre los que forman parte del conjunto respecto al resto del grafo. También existen **algoritmos misceláneos**, que no podemos llegar a clasificar en ninguna de los anteriores, como los basados en cadenas de Markov en tiempo continuo (CTMC).

2.3. Modelos de difusión

Con el objetivo de capturar las distintas formas en las que puede propagarse la influencia, varios autores han desarrollado distintos modelos de difusión D. Para nuestro problema, usaremos los 3 modelos más usados, siendo estos el modelo de Cascada Independiente, el modelo de Cascada Ponderada y Modelo de Umbral Lineal. Es notable indicar que los modelos de difusión surgieron a la vez que el problema computacional(Kempe et al. 2003).

Modelo de Cascada Independiente

El modelo IC (de la sigla del termino inglés *Independent Cascade Model*) es el más popular de todos ellos. Para los pesos de cada arista $w(u, v)$ será usada una constante, común para todas las aristas.

La difusión de la influencia comienza en el primer instante $i = 0$, donde solo se hallarán activos los nodos semilla $\mathcal{S}$. En cada iteración posterior $i > 0$, en cada nodo activo $u \in \mathcal{A}_{i-1}$ que haya sido activado en la iteración anterior se generará un número $rand \in [0, 1]$ aleatorio para cada uno de los nodos de su vecindad de salida $\mathcal{N}^{out}(u)$. Esto implicará que cada nodo activo $u \in \mathcal{A}_i$ solo tendrá una oportunidad de contagiar a sus vecinos de salida $v \in \mathcal{N}^{out}(u)$. Un nodo $u \notin \mathcal{A}_i$ pasará a ser activo $u \in \mathcal{A}_i$ si el número aleatorio $rand$ generado es mayor que el peso de su arista entrante $w(v, u)$. Este proceso continuará hasta que no se realice ninguna propagación, es decir, no se añada ningún nodo al conjunto de nodos activos $\mathcal{A}_n \in \varnothing$, siendo n el numero de iteraciones al final. De esta forma, el modelo tendrá garantía de acabar. Al tratarse de un modelo estocástico, el mismo conjunto de nodos semilla $\mathcal{S}$ puede conseguir resultados distintos, lo que nos obligará a realizar varias simulaciones para quedarnos con el caso medio. Se describe el pseudocódigo (Aghaee et al. 2021) en el algoritmo 1.

Algoritmo 1 Modelo de Cascada Independiente

Input: Grafo $G = (V, E, W)$, conjunto de nodos semilla $\mathcal{S}$
Output: Conjunto de nodos activados $\mathcal{A}_\infty$

```
i ← 0
while A_i ≠ ∅ do
    i ← i + 1
    A_i ← ∅
    for all v ∈ A_{i−1} do
        for all u ∈ N^out(v), u ∉ ⋃_{j=0}^{i} A_j do
            rand ← número aleatorio ∈ [0, 1]
            if rand < w(v, u) then
                Activa nodo u
                A_i ← A_i ∪ {u}
A_∞ ← ⋃_{j=0}^{i} A_j
return A_∞
```

Modelo de Cascada Ponderada

Este modelo WC (sigla del término en inglés *Weighted Cascade Model*) es muy similar al modelo IC descrito anteriormente, con la particularidad de que el peso de las aristas cambia. Para el peso de cada arista $w(u, v)$ se usará el inverso del grado de entrada del nodo v:

$$w(u, v) = \frac{1}{D(v)}$$

Esto implicará que un nodo aislado sea fácilmente influenciable por uno de sus vecinos de entrada $u \in \mathcal{N}^{in}(v)$, mientras que un nodo mucho más conectado sea más difícil de influenciar por un nodo en concreto, aunque habrá muchas más ocasiones de infectarlo al tener un gran número de vecinos $\mathcal{N}^{in}(u)$. Comparte el mismo pseudocódigo que el modelo IC (Véase algoritmo 1).

Modelo de Umbral Lineal

Para introducir el modelo LT (de su término en inglés Linear Threshold Model) necesitaremos otro conjunto de pesos $\theta(G) : V(G) \to [0, 1]$ será asignado a cada uno de los nodos. Cada uno de dichos valores representará el umbral de influencia de cada nodo, el cuál indicará que porcentaje de vecinos tienen que estar influenciados a su alrededor para que el nodo también sea influenciado. Cuanto mayor es su umbral de influencia, mayor cantidad de nodos influenciados a su alrededor harán falta para influenciar al nodo en cuestión.

Se partirá en el instante inicial $i = 0$ de la activación de los nodos semilla. En iteraciones posteriores $i > 0$, en lugar de tener una única oportunidad para infectar a los nodos de la vecindad de salida $\mathcal{N}^{out}(u)$, se comprobará para cada nodo no activo u de la vecindad de salida de cada nodo activo $u \notin \mathcal{A}_{i-1} \wedge v \in \mathcal{A}_{i-1} \wedge u \in \mathcal{N}^{out}(v)$ que su valor de umbral de influencia $\theta(u)$ es menor que la suma de las aristas entrantes pertenecientes a los nodos activos de la vecindad de entrada $\sum_{u \notin \mathcal{A}_{i-1} \wedge v \in \mathcal{A}_{i-1} \wedge u \in \mathcal{N}^{out}(v)} w(v, u) > \theta(u)$. En caso de ser así, este nodo pasará a formar parte de los nodos activos $\mathcal{A}_i$ y participará en la siguiente iteración de propagación de la influencia. De este modo se simula el comportamiento de ser influenciado por tu entorno si la mayor parte de él ha llegado a ser influenciado; en el caso de nodos más aislados solo harán falta unos pocos nodos mientras que en nodos mucho más conectados serán necesarios muchos más nodos infectados a su alrededor. El conjunto de valores de umbral de influencia $\theta(G)$ será asignado siguiendo una distribución de probabilidad p_θ y se hará como parte del procesado de datos independientemente a cada uno de los grafos. Esto implicará que el comportamiento de este modelo deje de ser estocástico, ya que, entre distintas simulaciones, aunque el umbral de influencia $\theta(G)$ haya sido asignado siguiendo la distribución de probabilidad, el valor del umbral de cada nodo no variará, por lo que para el mismo conjunto de nodo semilla obtendremos la misma propagación de la influencia. Los pesos $W(G)$ usados para las aristas en este modelo siguen el mismo patrón usado en el modelo WC anterior. El siguiente pseudocódigo (Aghaee et al. 2021) viene dado por el Algoritmo 2.

Algoritmo 2 Modelo de Umbral Lineal

Input: Grafo $G = (V, E, W)$, conjunto de umbrales de influencia $\theta(G)$, conjunto de nodos semilla $\mathcal{S}$

Output: Conjunto de nodos activados $\mathcal{A}_\infty$

$i \leftarrow 0$
while $\mathcal{A}_i \neq \varnothing$ **do**
    $i \leftarrow i + 1$
    $\mathcal{A}_i \leftarrow \varnothing$
    for all $v \in \mathcal{A}_{i-1}$ **do**
        for all $u \in \mathcal{N}^{out}(v), u \in \bigcup_{j=0}^{i} \mathcal{A}_j$ **do**
            if $\sum_{u \in \mathcal{N}^{in}(v), v \in \bigcup_{j=0}^{i} \mathcal{A}_j} w(u, v) \geq \theta(v)$ **then**
                Activa nodo v
                $\mathcal{A}_i \leftarrow \mathcal{A}_i \cup \{v\}$
$\mathcal{A}_\infty \leftarrow \bigcup_{j=0}^{i} \mathcal{A}_j$
return $\mathcal{A}_\infty$

3. Metaheurística de Colonia de Hormigas

La familia de metaheurísticas de colonia de hormigas se basa en el comportamiento natural de las hormigas en su búsqueda de comida, donde inicialmente recorren su entorno en busca de alimento, y a medida que lo van encontrando van depositando feromonas por su camino. Si en el recorrido una hormiga se encuentra con feromonas, tendrá una mayor tendencia a seguir ese rastro que otro sin estas, y así se van desarrollando distintos caminos para realizar la tarea con mayor eficacia. La feromona a su vez va evaporándose con el tiempo, lo cual ayuda que las hormigas no tomen siempre el mismo camino o se enfoquen demasiado en alguno. Esta metaheurística fue desarrollada por Marco Dorigo en 1992 en su tesis doctoral. Uno de los primeros problemas con el que fue usado fue el problema del viajante, donde debe buscarse el camino más corto entre un conjunto de ciudades.

3.1. Algoritmo básico

En todo algoritmo de colonia de hormigas, cada hormiga es un agente básico encargado de buscar una solución al problema, siendo este buscar el camino más corto en un grafo con pesos $G = (V, E, W)$. Para aplicar esta metaheurística al problema de maximización de la influencia, debe transformarse este objetivo de la hormiga a buscar aquellos nodos que consigan la mayor propagación de la influencia $\sigma(\mathcal{S})$ a través del uso del valor del fitness, que determinará en gran medida cómo se distribuyen las feromonas. Puede desglosarse el funcionamiento del algoritmo en 3 partes distinguibles:

- Generación de soluciones
- Actualización de feromonas
- Acciones *daemon*

A continuación hablaremos de cada una de ellas:

Generación de soluciones

Cada hormiga $h_z \in \mathcal{H}$ debe crear una solución $s(h_z)$ para el problema en cada iteración, donde comienza en un nodo de inicio, y va moviéndose por el grafo nodo a nodo, añadiéndolos a la solución $s(h_z)$. En la primera iteración no tendrá ninguna feromona que seguir así que lo hará de una manera totalmente aleatoria o usando algún algoritmo voraz, y a medida que se vaya acumulando feromona en los nodos del grafo, tenderá más a estas, aunque continuará eligiendo al azar siguiendo una distribución de probabilidad. Esta distribución de probabilidad de que una hormiga se mueva a un nodo en concreto la calculamos individualmente para cada nodo de la siguiente manera:

$$p(u) \leftarrow \frac{[\tau(u)]^{\alpha} \cdot [\eta(u)]^{\beta}}{\sum_{w \in V} [\tau(w)]^{\alpha} \cdot [\eta(w)]^{\beta}} \tag{3}$$

donde $\tau(u)$ es el nivel de feromona del nodo u y $\eta(u)$, la calidad del nodo u. Cuanto mayores sean estos dos valores, las hormigas serán más atraídas a incluir este nodo en su solución $s(h_z)$. Los valores $\alpha \in [0, 1]$ y $\beta \in [0, 1]$ son constantes que determinan la importancia del nivel de la feromona $\tau(u)$ y la calidad del nodo $\eta(u)$ respectivamente . La calidad del nodo es un valor heurístico calculado previamente, nosotros usaremos los propuestos por

los autores:

$$\eta(u) \leftarrow \rho \cdot \tau(u) \tag{4}$$

donde $\rho \in [0, 1]$ será el parámetro de evaporación, una constante que indicará cuánta feromona debe evaporarse en cada iteración del algoritmo. Para nuestro problema de maximización de la influencia, los nodos escogidos por cada hormiga $h_z \in \mathcal{H}$ irán en función del tamaño k de los nodos semilla $\mathcal{S}$ que servirán de solución. Por este motivo, ninguno de los k nodos puede estar repetido en la solución $s(h_z)$, así que cuando una hormiga visita un nodo, no vuelve a recorrerlo.

Actualización de feromonas

Una vez las hormigas $h_z \in \mathcal{H}$ han construido sus respectivas soluciones, es necesario cambiar el valor de las feromonas, depositando una cierta cantidad en cada nodo, para que la siguiente iteración se vean reflejados los resultados de la iteración actual. A su vez, también es necesario reducir el nivel de feromona en cada iteración simulando una evaporación, para poder evitar la convergencia prematura. De no existir este fenómeno de evaporación, las hormigas seleccionarían una y otra vez el mismo conjunto de nodos, y se acumularía la mayor cantidad de feromonas en este conjunto concreto, no dando oportunidad a soluciones más novedosas a aparecer, y consiguiendo quedarnos en un óptimo local. El valor de feromona en cada nodo se actualizará al final de cada iteración de la siguiente manera:

$$\tau(u) \leftarrow (1 - \rho) \cdot \tau(u) + \sum_{z=0}^{|\mathcal{H}|} \Delta\tau(u)^{h_z} \tag{5}$$

Donde $\triangle\tau(u)^{h_z}$ será:

$$\Delta\tau(u)^{h_z} = \begin{cases} f(s(h_z)) & \text{si } u \in s(h_z), \\ 0 & \text{en otro caso.} \end{cases} \tag{6}$$

Donde $f(s(h_z))$ es el fitness de la solución de la hormiga h_z. El fitness es una función normalmente usada en algoritmos genéticos que tiene como objetivo valorar la solución y determinar cuáles son mejores que otras. Sin esta función, no tendríamos forma de evaluar cómo se está llevando a cabo la búsqueda y si nos estamos acercando a la solución, haciendo que no sea mejor que una búsqueda aleatoria. El fitness debe de ser una función no muy costosa computacionalmente, ya que estaremos recurriendo a ella reiteradamente en cada iteración. De esta manera, si un nodo forma parte de alguna solución $s(h_z)$ de alguna hormiga, su valor de feromona se verá incrementado, mejorando sus probabilidades de ser escogido en siguientes iteraciones, mientras que si no es contenido en ninguna solución, su feromona se verá reducida y será más difícil que en el futuro alguna hormiga la incluya en su solución.

Acciones *daemon*

Aparte de generar las soluciones de cada hormiga, y posteriormente actualizar las feromonas de cada nodo, es posible que algunos algoritmos implementen de forma opcional ciertas optimizaciones que no entran dentro de ninguna de estas dos categorías. Su objetivo puede variar, pero por lo general se enfoca en realizar una pequeña búsqueda opcional para intentar mejorar las soluciones propuestas por el primer paso, con el objetivo de salir de un óptimo local en busca de la solución global.

A continuación se muestra el pseudocódigo (Kumar y Janga Reddy 2006) en el algoritmo 3, simplificado con el objetivo de servir de base para el resto de variantes de la metaheurística de colonia de hormigas para cualquier tipo de problema, no únicamente el de maximización de la influencia:

Algoritmo 3 Colonia de hormigas básico

Input: Grafo $G = (V, E, W)$

```
1: τ ← valor inicial
2: while Condición para acabar no alcanzada do
3:     for h_z ∈ ℋ do
4:         while h_z no haya completado su recorrido do
5:             s(h_z) ← s(h_z) ∪ {nodo elegido}
6: Realizar acciones daemon
7: Actualizar los valores de feromona τ
```

3.2. Algoritmo propuesto por Singh et al.

Partiendo de esta base para el algoritmo de la metaheurística de colonia de hormigas, Singh et al. 2020 proponen el Influence Maximization algorithm based on Ant Colony Optimization (ACO-IM). En el algoritmo 4 se muestra el pseudocódigo (Singh et al. 2020) con todas las implementaciones desarrolladas, que iremos analizando una a una.

Valor de difusión esperado

Para el fitness, calculado en las lineas 13 y 37, necesitamos una función poco costosa computacionalmente, pudiendo usar el valor de difusión esperado para ello. El EDV (por su término en inglés *Expected Diffusion Value*) tiene como objetivo calcular la propagación de la influencia $\sigma(\mathcal{S})$ de un conjunto de nodos semilla $\mathcal{S}$ bajo el modelo IC, sin realizar ningún tipo de simulación. Para ello, se comprueba la influencia que tiene cada nodo $u \in \mathcal{S}$ sobre los nodos de su vecindad de salida $\mathcal{N}^{out}(u)$ y se realiza una estimación de cuántos nodos en una situación promedio deberían ser influenciados. Denominaremos como $N^{(1)}(\mathcal{S})$ al conjunto de vecinos de los nodos semilla, indicando el número cuántos saltos consideramos como vecino (hasta ahora normalmente era 1, pero podría llegar a ser más). Se calcula de la siguiente manera:

$$\mathrm{EDV}(\mathcal{S}) = \sigma_0(\mathcal{S}) + \sigma_1(\mathcal{S}) = k + \sum_{u \in \mathcal{S}} (1 - \prod_{(u,v) \in E, v \in N^{(1)}(\mathcal{S}) \setminus \mathcal{S}} (1 - p_{uv})) \tag{7}$$

El inconveniente de este enfoque es que no se está teniendo en cuenta que pasaría con el modelo LT (con el modelo WC no habría ningún problema, ya que solo cambia el peso de las aristas y eso está incluido en la fórmula). Por ello, Singh et al. 2020 proponen un EDV modificado, que tendrá en consideración los umbrales de influencia $\theta(G)$ de los nodos, además de que no solo tendrá en consideración los nodos de la vecindad de salida $\mathcal{N}^{out}(u)$ de cada nodo $u \in S$, sino que calcula la propagación de la influencia $\sigma(\mathcal{S})$ en dos saltos, es decir, comprobando además los vecinos salientes v de la vecindad de salida de cada nodo semilla $u \in S, v \in \mathcal{N}^{out}(\mathcal{N}^{out}(u))$. Aún así, en nuestro trabajo seguiremos usando la formula original de EDV de un solo salto, ya que los resultados obtenidos entre uno y otro no varían demasiado, y computacionalmente es menos costoso calcular la primera de ellas.

Algoritmo 4 ACO-IM(G,k)

Input: Grafo $G(V, E, W, \theta)$, tamaño semilla k.
Output: Conjunto de semillas $\mathcal{S}$.

$i \leftarrow 0$ ▷ *Fase de inicialización*
$\forall u \in V(G), \tau_0(u) \leftarrow 0$
$S_{mejor} \leftarrow \varnothing$
$s(h_z) \leftarrow \varnothing; |s(h_z)| = k; \forall h_z \in \mathcal{H}$
Distribución de hormigas por el grafo ▷ *Inicialización de hormigas*
- $R_G \leftarrow$ Ordena cada nodo $u \in V(G)$ en orden descendente según su grado
- Distribuye las hormigas en los nodos de R_G

Inicializar la solución para cada hormiga $h_z \in \mathcal{H}$ ▷ *Inicialización de soluciones*
for each $h_z \in \mathcal{H}$ en los primeros *nants* nodos $u \in R_G$ **do**
    $s(h_z) \leftarrow u$
    $s(h_z) \leftarrow s(h_z) \cup$ Elegir $(k-1)$ primeros nodos de $R_G \setminus u$
    for each $v \in \{s(h_z) \setminus u\}$ **do**
        if $rand > p_{reemp}$ **then**
            $v \leftarrow Replace(v)$
$f(s(h_z)) \leftarrow$ Calcular fitness para cada hormiga $h_z \in \mathcal{H}$ ▷ *Cálculo de fitness*
Ordenar hormigas según el fitness $f(s(h_z))$ de su solución $s(h_z)$
$S_{mejor} \leftarrow$ Selecciona mejor $p\,\%$ soluciones según su fitness
Inicializar feromona en cada nodo $u \in V(G)$ ▷ *Inicialización de feromonas*
for each solución $s(h_z) \in S_{mejor}$ **do**
    for each nodo $u \in s(h_z)$ **do**
        $\tau_0(u) \leftarrow \tau_0(u) + f(s(h_z))$

while $i < I_{max}$ **do** ▷ *Fase de búsqueda de soluciones*
    $i \leftarrow i + 1$
    for each ant $h_z \in \mathcal{H}$ **do**
        $s(h_z) \leftarrow \varnothing$
        $j \leftarrow 1$
        while $j \leq k$ **do**
            $rand \leftarrow$ número aleatorio entre 0 y 1
            if $rand > p_i$ **then** ▷ *Convergencia prematura*
                $s \leftarrow$ Seleccionar aleatoriamente nodo $N \in V$
            else
                $rand \leftarrow$ número aleatorio entre 0 y 1
                if $rand < r_0$ **then** ▷ *Generación de soluciones*
                    $s \leftarrow$ Seleccionar nodo con Ecuación 9
                else
                    $s \leftarrow$ Seleccionar nodo con Ecuación 3
            $s(h_z) \leftarrow s(h_z) \cup s$
            $j \leftarrow j + 1$
        Calcular $f(s(h_z))$ ▷ *Cálculo de fitness*
    Ordenar hormigas según el fitness $f(s(h_z))$ de su solución $s(h_z)$
    $S_{mejor} \leftarrow$ Selecciona mejor $p\,\%$ soluciones según su fitness
    for each solución $s(h_z) \in S_{mejor}$ **do**
        for each nodo $u \in s(h_z)$ **do**
            Actualizar $\tau(u)$ con Ecuación 11 ▷ *Actualización de fitness*
return $argmax_{\forall h_z \in \mathcal{H}} \sigma(s(h_z))$

Mecanismo contra la convergencia prematura

Para evitar que el algoritmo se quede atrapado en locales óptimos en las primeras iteraciones del algoritmo y que las hormigas propongan soluciones que no se acercan al óptimo global, proponen un parámetro probabilístico $p_i \in [0, 1]$, calculado en la linea 27 de la siguiente manera:

$$p_i = \frac{\log(i)}{\log(I_{max})} \tag{8}$$

donde i es la iteración actual e I_{max} el número máximo de iteraciones. Al principio, p_i valdrá 0, y a medida que vayan aumentando las iteraciones, irá poco a poco aumentando hasta 1. En cada iteración, cuando cada hormiga vaya a escoger a un nodo que añadir a su solución, antes generará un número aleatorio $rand \in [0, 1]$. En caso de que este sea mayor que p_i, en vez de escoger un nodo siguiendo la distribución de probabilidad, escogeremos un nodo al azar de todo el grafo que no sea ya parte de la solución. De esta manera, en las primeras iteraciones obtendremos soluciones más dispares, y así poder conseguir no quedarnos en un óptimo local al principio, mientras que hacia al final, ya no escogeremos nodos de esta manera para dejarse guiar más por el resultado de las feromonas.

Mecanismo contra el estancamiento

Es posible que las hormigas empiecen a generar soluciones similares o incluso idénticas entre ellas. Esto no es deseable, ya que reduce sustancialmente el campo de búsqueda de la solución óptima. El estancamiento es gravemente influenciado por el parámetro de evaporación ρ, que en caso de ser muy alto , reduce la diversidad de niveles de feromonas entre los nodos, al evaporarla con demasiada rapidez. La feromona depositada en cierta iteración, con un parámetro de evaporación alto, perdura muy pocas interacciones y acaba igualándose el nivel de feromona de todos los nodos. Para evitarlo, debemos establecer el parámetro de evaporación a un nivel bajo, establecido por Singh et al. 2020 a $\rho = 0{,}2$, y utilizar los otros dos parámetros α y β para dar cierta relevancia a la feromona $\tau(u)$ del nodo y a su importancia $\eta(u)$. Se establecerán estos parámetros a $\alpha = 0{,}5$ y $\beta = 0{,}5$, también indicados por los autores de ACO-IM.

Modificación en la generación de soluciones

En todo algoritmo de colonia de hormigas es necesario inicializar las feromonas, y para ello Singh et al. 2020 proponen generar primero una serie de soluciones iniciales con tal de tener material para establecer valores relevantes de feromonas por el grafo. Para ello, calcularemos cuál es el conjunto R_G en la linea 5, siendo estos los nodos que tienen un grado D^{out} mayor que el de la media de nodos. Con ellos identificados, se generará una hormiga por cada nodo $u \in R_G$, para que lo tenga como nodo inicial, y construirá su solución escogiendo los $k - 1$ nodos $u \in R_G$ con mayor grado D^{out}. Esto crearía gran cantidad de soluciones muy similares (las primeras k hormigas tendrán un solución idéntica), que solo se tendrán en cuenta para la inicialización de las feromonas, pero influenciarían mucho en como se desarrollaría el resto del algoritmo y ayudaría mucho al estancamiento de soluciones. Para ello, al tener creadas estas soluciones iniciales, por cada nodo de la solución que no sea donde la hormiga tiene el nodo de inicio, se generará un número aleatorio $rand \in [0, 1]$ en la linea 11, y si es menor que una constante p_{reemp}, se intercambiará ese nodo por uno al azar del grafo, consiguiendo soluciones mucho más dispares.

Además, Singh et al. 2020 proponen un segundo enfoque propio de una variante de

la metaheurística de colonia de hormigas llamado *Ant Colony System* para la generación de soluciones. Una vez se realiza la evasión de convergencia prematura, se realizará la elección entre añadir el nuevo nodo siguiendo la distribución de probabilidad o a través de este nuevo método. Con la finalidad de poder escoger, existirá un parámetro $r_0 \in [0, 1]$ que indicará la probabilidad asignada a elegir un método o otro. Se generará un número aleatorio $rand \in [0, 1]$, y en caso de ser menor que r_0 se escogerá el nuevo método de la linea 32, y en otro caso seguiremos con el que teníamos en la linea 34. Este nuevo método consistirá en escoger el nodo más prometedor que no exista dentro de la solución de la hormiga. Para ello se tendrá en cuenta los mismos factores que hacían más o menos probable la selección de un nodo en la distribución de probabilidad N: la feromona del nodo y su correspondiente importancia. La forma de escoger el nodo v se expone a continuación:

$$v \leftarrow \begin{cases} argmax_{v \in V \setminus x}[\tau(v)]^{\alpha} \cdot [\eta(v)]^{\beta} & \text{si } r < r_0 \\ v \text{ siguiendo la distribución de probabilidad} & \text{en otro caso.} \end{cases} \tag{9}$$

Este enfoque es mucho más agresivo que el anterior implementado, ya que selecciona aquel nodo más prometedor, sin dar tanto pie a la aleatoriedad que hace característico a las metaheurísticas. Por una parte es algo beneficioso ya que dará lugar a escoger mejores nodos para las soluciones, mientras que también es una desventaja, ya que estaremos aportando al estancamiento, creando soluciones que se parecerán mucho más entre ellas, ya que el mejor nodo para una hormiga también lo va a ser para otra (siempre y cuando no lo tenga ya). Esto puede regularse en función de la r_0, dándole más o menos relevancia al nuevo enfoque agresivo.

Modificación en la actualización de feromonas

Aparte de la variación en la generación de soluciones, también implementa algún cambio en la actualización de feromonas. Se introducirá el concepto del conjunto de mejores soluciones S_{mejor}, donde irán incluidas el $p_{top} \in [0, 1]$ porcentaje de mejores soluciones de la iteración actual. Se valorarán en función de su fitness, a mayor sea este, más probabilidad tendrán de estar en este conjunto. Serán las soluciones de estas hormigas, y no de otras, las que se encargarán tanto de inicializar la feromona en la iteración inicial $i = 0$, como de ir depositando en futuras iteraciones $i > 0$. En la primera iteración, para inicializar las feromonas, en vez de asignar el valor de feromona de todos los nodos a un valor inicial τ_0, se calcula el conjunto S_{mejor} para que pueda depositar en la linea 42 en los nodos $u \in s(h_z) \in S_{mejor}$ una cantidad de feromona igual a la suma del fitness de las soluciones que contienen a ese nodo u:

$$\tau(u) \leftarrow \tau_0 + \sum_{z=0}^{|S_{mejor}|} \Delta\tau(u)^{s(h_z)} \tag{10}$$

donde $\Delta\tau(u)^{s(h_z)}$ será el análogo a la ecuación 6.

En posteriores iteraciones, también es alterado la forma de actualizar la feromona. Como en la inicialización, también será el conjunto de hormigas cuyas soluciones se encuentra en S_{mejor} las que se encarguen de depositar la feromona, pero también habrá otra sutil modificación. Las hormigas ya no dejarán una feromona con valor igual al fitness, sino que se verá reducida de forma constante por el parámetro de evaporación y de forma gradual por el número de iteraciones: A más iteraciones pasen, menor cantidad de feromona irán

dejando. El cálculo de la feromona entonces se hará de la siguiente manera:

$$\tau(u) \leftarrow (1-\rho)\cdot\tau(u) + \frac{\rho}{i}\cdot\sum_{z=0}^{|S_{mejor}|}\Delta\tau(u)^{s_{h_z}} \tag{11}$$

donde $\triangle\tau(u)^{s(h_z)}$ seguirá siendo la fórmula vista en la ecuación 6.

4. Propuesta de mejora

Una vez analizado a fondo el algoritmo ACO-IM Singh et al. 2020 podemos tratar de probar nuestra hipótesis: Utilizar una forma alternativa de actualizar las feromonas con el fin de ver diferencias notables en la ejecución de ACO-IM. Además, veremos las distintas decisiones de diseño que se han tomado a la hora de implementar los algoritmos, por cuestiones de optimización temporal o de eficiencia.

4.1. Mecanismos de actualización de las feromonas

Existen una gran cantidad de variantes, cada una con ideas nuevas que aportar, ya sea en la generación de soluciones o la actualización de feromonas. Nos centraremos en estudiar 3 de las variantes más prominentes (Pérez-Carabaza et al. 2022): El Rank-based Ant System, el Max-Min Ant System y el Ant Colony System. Usaremos las versionas introducidas en el artículo sin las extensiones propuestas posteriormente. Las veremos más a fondo a continuación:

Rank-based Ant System (ASRank)

Esta extensión de la metaheurística de colonia de hormigas propuesta por Bullnheimer et al. 1999 consiste en establecer un ranking entre las hormigas, donde cuanto mejor fitness tenga en su solución, más alto se encontrará en el ranking. Usaremos este ranking con el objetivo de realizar una actualización de feromonas proporcional, donde aquellas hormigas con mejor clasificación en el ranking dejarán una cantidad mayor de feromona. Será similar a lo visto en ACO-IM, teniendo un conjunto de mejores soluciones y solo permitiendo a ese al $p\,\%$ de hormigas depositar feromona en los nodos. Además, se llevará la cuenta de la hormiga con mejor fitness h_{mejor} hasta la fecha, en vez de fijarse solo en la mejor hormiga de la iteración, donde será esta mejor hormiga la que más feromona depositará de todas. La fórmula para la actualización de feromonas cambiará por lo tanto a la siguiente:

$$\tau(u) = (1-\rho)\cdot\tau(u) + \sum_{r=1}^{w-1}\Delta\tau(u)^{r} + w\cdot\Delta\tau(u)^{h_{mejor}} \tag{12}$$

Donde w sera el número de hormigas que depositen feromona, que puede ir en función de un porcentaje de hormigas como en ACO-IM o ser una constante de hormigas fija, y r indicará el puesto en el ranking de cada hormiga. $\Delta\tau(u)^{s_i}$ seguirá refiriéndose a la ecuación 6.

Max-Min Ant System (Max-Min)

Esta variante fue creada por Stützle y Hoos 2000, y se centra en la idea de mantener unos umbrales máximos y mínimos para el valor de las feromonas. Esta feromona solo se

actualizará usando la mejor hormiga h_{mejor} de la iteración o de todas las iteraciones, en función de como se establezca (En nuestro caso se ha optado por la mejor hormiga de la iteración). Se expone la formula de actualización a continuación:

$$\tau_i = (1 - \rho)\tau_i + \Delta\tau_i^{h_{mejor}} \tag{13}$$

Para establecer los umbrales, como máximo se propone $\tau_{max} = \frac{f(h_{mejor})}{\rho}$ y como mínimo $\tau_{min} = \frac{\tau_{max}(1-\sqrt[n]{0{,}05})}{(0{,}5n-1)}$, donde n es el número de nodos. En cada iteración del algoritmo, debe de ser comprobado que $\forall u \in V, \tau_{min} < \tau_i(u) < \tau_{max}$, y de no ser el caso, se establecerá el valor de la feromona al umbral mínimo o máximo en función de qué límite haya sobrepasado. $\Delta\tau(u)^{h_{mejor}}$ sigue haciendo referencia a la ecuación 6.

Ant Colony System (ACS)

En esta extensión, sugerida por Dorigo y Gambardella 1997, se introdujo la idea anteriormente mencionada en ACO-IM de realizar una búsqueda algo más agresiva a la hora de crear soluciones para las hormigas. A su vez, también utiliza el concepto de únicamente usar la mejor hormiga h_{mejor} hasta la fecha para actualizar todas las feromonas, siguiendo la misma formula 13. Además de estos cambios, también propone actualizar la feromona mientras se genera cada solución; al visitar un nodo, la feromona de este nodo cambiará en ese mismo instante (afectando a esa misma iteración en vez de tener a la siguiente al esperar al final de la iteración para actualizar las feromonas). La fórmula con la que se actualizará en ese momento su feromona será con:

$$\tau_i = (1 - \zeta)\tau i + \zeta\tau_0 \tag{14}$$

Donde τ_0 es el valor inicial de la feromona y $\zeta \in [0, 1]$ es una constante que indicará la relevancia que le daremos al valor actual de la feromona frente al valor inicial que tenia al principio del algoritmo.

4.2. Decisiones de diseño e implementación

En esta sección hablaremos más a fondo sobre como han sido implementados la serie de algoritmos vistos hasta ahora, y que decisiones de diseño se han tomado con el fin de que los algoritmos rindan mejor o tarden menos tiempo.

Lectura de ficheros

A la hora de leer los grafos sacados de Patwardhan et al. 2022, (cuya mayoría de links están caídos, pero pueden recuperarse gracias a la ayuda de páginas que mantienen un archivo de páginas como `web.archive.org`), los formatos de los grafos son dispares, variando desde simples ficheros de texto ".txt", otras extensiones como ".csv", ".mtx.º ".gml", hasta archivos sin formato. Por suerte, la librería networkx implementa la lectura de ficheros ".gml", pero el resto de formatos no son directamente soportados, así que se ha optado por implementar la lectura de ficheros de forma general para todos, ignorando las cabeceras que crean las diferentes extensiones. Algunos grafos serán no dirigidos, mientras que otros sí serán dirigidos. Como nuestro algoritmo le interesan que los grafos sean dirigidos, los trataremos a todos como tal, y en caso de los no dirigidos será necesario añadir la arista inversa por cada arista presente en el grafo. Posteriormente, se convierte

el nombre de todos los nodos a enteros, empezando desde 0, para que la nomenclatura entre los nodos no sea tan dispar, ya que no nos interesa que el grafo contenga nombres o números no consecutivos.

Asignación de pesos

Los grafos, al ser leídos, carecen de ningún peso asignado a sus aristas (Y en caso de que el formato de su fichero lo tuviere, es ignorado). Por ello, en caso de los modelos WC y LT, simplemente debe asignarse a cada arista saliente de cada nodo u el peso correspondiente, siendo el inverso del grado de salida del nodo $\frac{1}{D^{out}(u)}$. Por otra parte, en el modelo IC, al venir marcado el peso de las aristas por el umbral de percolación, se extraerá este de los datos para ser asignado.

Creación de umbrales de influencia

Anteriormente se mencionó que los umbrales de influencia $\theta \in [0, 1]$ eran previamente creados, siguiendo una distribución de probabilidad uniforme. Esto es porque necesitamos que entre distintas ejecuciones con distintos modelos, o al volver a hacer más pruebas con el mismo grafo, los umbrales de influencia se mantengan para que no haya variaciones posteriormente en los resultados. Para ello, se generan y almacenan en ficheros, siguiendo la nomenclatura anteriormente descrita de asignar enteros a cada nodo, y posteriormente se asignan a cada nodo.

Definición de hormiga

Cada hormiga $h_z \in \mathcal{H}$ se comportará como un agente simple, y por eso su implementación será bastante básica. La clase hormiga contará con un conjunto solución, con el objetivo de almacenar los nodos que formen parte de la actual solución, un atributo k que le indicará de que tamaño debe de ser la solución $s(h_z)$ y otro atributo de nodo de inicio, que marcará donde inicia la hormiga la primera solución (Aunque no se usará en posteriores soluciones, ya que estás dependerán de las feromonas). También contiene un atributo fitness $f(s(h_z))$ para almacenar el fitness de la solución actual encontrada. Dispone a su vez de los métodos necesarios para modificar la solución, fitness y nodo de inicio, donde los dos primeros los necesitaremos en cada iteración, mientras que el último solo será en la inicialización de feromonas.

Selección de nodos

A la hora de escoger un nodo para añadirlo a la solución tenemos tres maneras distintas:

- Seleccionar un nodo completamente al azar si $rand > p_i$ en función de la ecuación 8.
- Escoger el nodo v con mayor $[\tau(v)]^{\alpha} * [\eta(v)]^{\beta}$ siguiendo la ecuación 9.
- Coger un nodo partiendo de la distribución de probabilidad N descrita por la ecuación 3.

Para los dos primeros casos no tenemos problema (siempre respetando que el nodo $u \notin s(h_z)$), y su complejidad computacional no es muy alta, aunque es cierto que para el

segundo caso debemos previamente en cada iteración ordenar los nodos en función de la feromona e importancia: $[\tau(v)]^{\alpha} * [\eta(v)]^{\beta}$, lo cual implicará un $O(n \log n)$.

Para el tercer caso se nos complica un poco más, ya que debemos haber calculado la probabilidad p de cada nodo (que también debemos hacerlo en cada iteración) y luego seleccionar en función de dicha probabilidad. En vez de hacerlo iterativamente, donde tendríamos que generar un número aleatorio $rand \in [0, 1]$, e ir recorriendo en el peor de los casos todos los nodos del grafo para sumar todas las probabilidades y quedarnos con aquel nodo donde la probabilidad que hayamos ido sumando sea mayor que el número aleatorio $rand$, se ha decidido hacerlo con otro enfoque. En vez de asignar a cada nodo su probabilidad, le asignaremos la probabilidad acumulada, siendo esta la suma de todas las probabilidades ya asignadas hasta el momento. De esta forma, si ordenamos los nodos en función de su probabilidad acumulada, podemos seleccionar de manera mucho más eficiente en una lista ordenada usando una búsqueda binaria, que encuentre aquel nodo con una probabilidad acumulada p_{acum} menor que supere el número $rand$ (comprobando que la probabilidad acumulada del anterior no sea superior que el número $rand$). De esta manera, reduciremos de una complejidad temporal de $O(n)$ a una de $O(\log_2 n)$. No habrá el problema de añadir complejidad temporal al ordenar los nodos, ya que siempre se recorren siguiendo la nomenclatura de enteros anterior, de menor a mayor, y así estarán asignadas las probabilidades acumuladas, así que nos ahorraremos una ordenación de $O(n \log n)$.

5. Experimentación y resultados

En este capítulo trataremos el tema de cómo se han realizado las pruebas, y se comentarán los resultados obtenidos.

5.1. Objetivo

Hemos partido del algoritmo de metaheurística de colonia más básico, y hemos visto tanto la variante ACO-IM como las tres distintas extensiones ASRank, Max-Min y ACS. Nuestra meta ahora es ver cómo se desenvuelven en la tarea del problema de la maximización de la influencia, y compararlos en función de la propagación de la influencia $\sigma(\mathcal{S})$ conseguida por cada una de las soluciones encontradas por estos algoritmos, tanto en las redes reales como en las artificiales de Erdős-Rényi.

5.2. Pesos de aristas usados

Necesitaremos escoger un peso para las aristas, para el modelo de difusión IC. En caso de que escogiésemos un valor muy grande para la probabilidad de las aristas, el problema de la maximización de la influencia sería trivial, ya que infectando cualquier conjunto de nodos conseguiríamos una propagación de la influencia muy grande al ser muy probable que un nodo infecte a otro. En el caso contrario, si escogiésemos un valor muy pequeño para la probabilidad de las aristas, por muy bien que escogiésemos aquellos nodos que formarán parte de nuestros nodos semilla $\mathcal{S}$ para inicializar la propagación de la influencia, conseguiríamos propagaciones muy aisladas, ya que sería ínfima la probabilidad de que un nodo influencie a otro. Por eso se introduce el término de umbral de percolación p^*, que indica, explicado a grandes rasgos, cuál es el punto medio qué conseguiríamos entre una infección muy localizada y una propagación masiva. Sin embargo, calcular este valor es una tarea muy complicada que se saldría del campo de estudio de este TFG, por lo tanto,

partiremos de grafos cuyo umbral de percolación p^* ya ha sido calculado previamente (Patwardhan et al. 2022).

5.3. Grafos Erdős-Rényi

Con el objetivo de tener un mayor número de grafos qué analizar y poder sacar conclusiones, también haremos uso de grafos sintéticos cuyo valor de umbral de percolación p^* sea más fácil de calcular. Para ello introduciremos los grafos de Erdős-Rényi. Este modelo de grafos fue creado por Erdös y Rényi 1959, de los cuales toman su nombre. Se tratan de grafos completamente aleatorios, donde partiendo de un número establecido de nodos, podemos darle un número m de aristas que se repartan por la red o una probabilidad $p \in [0, 1]$ para que cada arista se vea representada o no en el grafo de forma independiente. Para nuestro estudio se usará el segundo modelo. Al interesarnos que haya una única componente conexa, debemos escoger una probabilidad p lo suficientemente alta para que haya el suficiente número de aristas y todos los nodos estén conectados entre sí. El motivo de que nos interese este tipo de grafos es que, siempre y cuando le tratemos como un grafo no dirigido, su umbral de percolación p^* será igual al inverso del grado medio: $p^* = \frac{1}{\overline{D(G)}}$. Como nosotros trataremos con grafos dirigidos, deberemos añadir la arista inversa de cada arista del grafo con el objetivo de que el umbral de percolación p^* no varíe. Desgraciadamente, a pesar la ventaja de que podamos calcular fácilmente el umbral de percolación p^*, estos grafos no son ideales para modelar nuestro entorno, ya que tienen muy poco agrupamiento, es decir, no se forman grupos diferenciales entre los nodos. Aun así, eso no evita que no sean extremadamente útiles para tener un mayor número de grafos de prueba.

Para la creación de estos grafos artificiales, se ha optado por generar una serie de ellos con una variación de cantidad de nodos desde $N = 100$ a $N = 1000$, en saltos de 100. También será necesario asignarles una probabilidad p de generación de aristas. Para que no hubiesen demasiadas aristas, pero el grafo fuese completamente conexo, se ha optado por un $p \in 0{,}04, 0{,}06, 0{,}08, 0{,}01$. Esto nos generará un conjunto de 40 grafos distintos para experimentación. La librería networkx contiene un método para la generación de este tipo de grafos, que posteriormente guardaremos en el formato ".gml"que trae la librería para su fácil manipulación.

5.4. Métricas usadas

Necesitamos alguna forma de ser capaces de ver cómo rinde cada algoritmo en cada modelo de difusión distinto para todos los grafos que disponemos. Al principio se optó por conseguir el módulo de la propagación de la influencia $\sigma(\mathcal{S})$ por los nodos influenciados $u \in S$ en cada simulación para cada grafo, pero no podemos comparar usando distintos grafos ya que tienen tamaños variables. Por ello, usaremos una medida basada en (Erkol et al. 2019) para comparar entre los distintos algoritmos, siguiendo usando la propagación de la influencia $\sigma(\mathcal{S})$ pero interpretada de distinta manera. Realizaremos una serie de 11 pruebas con cada algoritmo, donde variaremos el tamaño k de la solución $\mathcal{S}$ para obtener soluciones de distinto tamaño (Cuanto más grande la solución, debería conseguir una propagación de la influencia mayor). La definiremos como S_k. Una vez encontremos las 11 distintas soluciones S_k, ejecutaremos una serie de 500 simulaciones para calcular la propagación de la influencia $\sigma(\mathcal{S})$ en cada una de ellas, a la que denominaremos O^k_{alg}, donde alg será el nombre del algoritmo que estemos procesando. Realizaremos la suma de las 11 de la

Símbolo	Valor	Significado
k	Varia	Tamaño de la solución
$\tau_0(u)$	0	Feromona inicial
α	0.5	Importancia de la feromona
β	0.5	Importancia de la calidad del nodo
ρ	0.2	Parámetro de evaporación
I_{max}	100	Número máximo de iteraciones
p_{top}	0.25	Porcentaje de hormigas que depositan feromona
p_{reemp}	0.5	Probabilidad de reemplazo de un nodo
r_0	0.5	Probabilidad de usar una formula 9 o 3
n_{horm}	20	Número de hormigas buscando soluciones

Tabla 1: Parámetros usados.

siguiente manera:

$$A_{alg} = \sum_{k=1}^{11} O_{alg}^{(r_k)} \tag{15}$$

donde $r_k = \lfloor [0{,}01 + (k-1)0{,}004]N \rfloor$. De esta manera variaremos la k en función del tamaño del grafo, yendo desde un número de nodos semillas del 1 % al 5 %, con pequeños incrementos. Por último, con el fin de comparar entre los distintos algoritmos, sacaremos el A_{alg} de cada uno de ellos, y calcularemos el ratio R_{alg} entre ellos:

$$R_{alg} = \frac{A_{alg}}{A_{base}} \tag{16}$$

donde A_{base} será la medida A_{alg} calculada para el algoritmo básico. Con esta medida estamos consiguiendo saber cuántas veces mejor es un algoritmo respecto al básico en ese grafo, y podremos usar esta medida para ver que algoritmo lo está haciendo mejor. Repetiremos estás medidas para cada modelo, en cada uno de los grafos distintos de los que disponemos. Para representar los datos, dispondremos de una serie de diagramas de caja para los grafos reales, uno por cada modelo de difusión distinto (ya que no tiene sentido comparar entre distintos modelos de difusión), y otra serie de diagramas de caja para los grafos Erdős-Rényi. En cada diagrama de caja habrá 4 cajas, una por cada variante de la metaheurística de colonia de hormigas que vamos a comparar. Debe indicarse que para la comparación de los algoritmos, se han desechado los valores atípicos en los diagramas de caja, es decir, no aparecen aquellos datos que lo hacen extremadamente bien o mal que sean puntuales, ya que provocan que las gráficas se aprecien peor, y las cajas se achaten.

5.5. Parámetros usados

Toda la implementación se ve afectada por el valor de los distintos parámetros usados. La selección de estos valores viene dada por la implementada en el algoritmo ACO-IM del artículo (Singh et al. 2020). En la tabla 1 podrá verse una recopilación de los parámetros presentes en los algoritmos, así como su valor y significado.

Nota: El número de hormigas en el artículo venia dado por la cantidad de nodos $u \in R_G$ que tenían un grado $D(u)$ mayor que la media. Esto ha demostrado ser un número de hormigas muy extenso, por lo que se ha reducido a una constante para poder hacer ejecuciones en un tiempo razonable, ya que la complejidad computacional depende directamente del número de hormigas. El resto de constantes vienen sacadas del articulo (Singh

	IC		WC		LT	
	$\overline{A_{alg}}$	Rango	$\overline{A_{alg}}$	Rango	$\overline{A_{alg}}$	Rango
Basico	2909.72	4831.74	11514.43	24187.22	17432.31	46727.0
ACO-IM	3036.48	5065.80	11866.05	25278.69	17903.40	47472.0
Rank-Based	3071.87	4882.89	12373.00	25659.64	18813.0	48446.0
Max-Min	3021.49	4898.69	11952.11	25220.19	18016.04	46547.0
Colony	3142.37	4954.74	12606.50	25381.59	19101.31	46785.0

Tabla 2: A_{Alg} conseguido en grafos reales

et al. 2020), a excepción de el número máximo de iteraciones I_{max} que es arbitrario, la feromona inicial $\tau_0(u)$ que simplemente se usa para inicializar y el tamaño de la solución k que varia en función de como estemos haciendo las pruebas, que veremos en el apartado de experimentación.

5.6. Entorno de simulación

Todas las pruebas han sido realizadas usando un ordenador Windows de 64 bits, con un procesador Intel(R) Core(TM) i7-9750H @ 2.60GHz 2.59 GHz de 16 GB de RAM. Para realizar las pruebas, se han realizado 500 simulaciones independientes en los 20 grafos reales, y los 40 grafos artificiales de Erdős-Rényi, disponibles en el Anexo II.

5.7. Resultados obtenidos en grafos reales

En primer lugar comentaremos los resultados obtenidos para grafos reales, y posteriormente comentaremos que hemos obtenidos con los grafos de Erdős-Rényi. Para empezar mostraremos en la tabla 2 todos los resultados A_{alg} medios conseguidos junto a su rango en los grafos reales por cada modelo de difusión estudiado:

Es posible apreciar como en la tabla 2 hay una gran diferencia entre el rendimiento de los distintos modelos de difusión, siendo el modelo IC el que menor A_{alg} está consiguiendo, y el modelo LT la mayor A_{alg}. Entre los modelos IC y WC es fácilmente explicable la diferencia, ya que conseguiremos una mayor propagación de la influencia $\sigma(\mathcal{S})$ cuanto mayor sean los pesos de probabilidad de difusión $W(G)$. Al usar el modelo IC el valor de percolación, que es del orden de 10^{-2} por lo general, es mucho menor en promedio que el valor usado en el modelo WC al usar el inverso del grado de entrada $D^{in}(v)$. En cuanto a comparar los modelos WC y LT que comparten los pesos de las aristas, entra en juego la facilidad que existe al influenciar a un nodo en un modelo u otro. Mientras que en el modelo WC solo se tiene una oportunidad de influenciar a un nodo desde otro en concreto, el modelo LT tiene tantas oportunidades como iteraciones existan con el nodo influenciado, lo que no solo hace que no sea estocástico sino que consigue además mejores resultados.

Debe hacerse notar que para poder comparar con mayor facilidad las gráficas, se han establecido como límites en el eje Y ciertas constantes (mínimo de 0.89 y máximo de 1.23).

En la figura 1 podemos ver un rendimiento parecido entre los 4 algoritmos, aunque puede observarse que todos se encuentran por encima del básico (es decir, que su $A_{alg} > 1$), exceptuando el algoritmo Max-Min que en algunos grafos lo hace algo peor. La variabilidad en el algoritmo ACO-IM es bastante baja, es decir, lo suele hacer igual de bien para todos los grafos, pero tanto los algoritmos de Rank-Based y Colony, aunque su variabilidad sea algo más alta, consiguen mejores resultados en general, siendo el último de ellos el que se

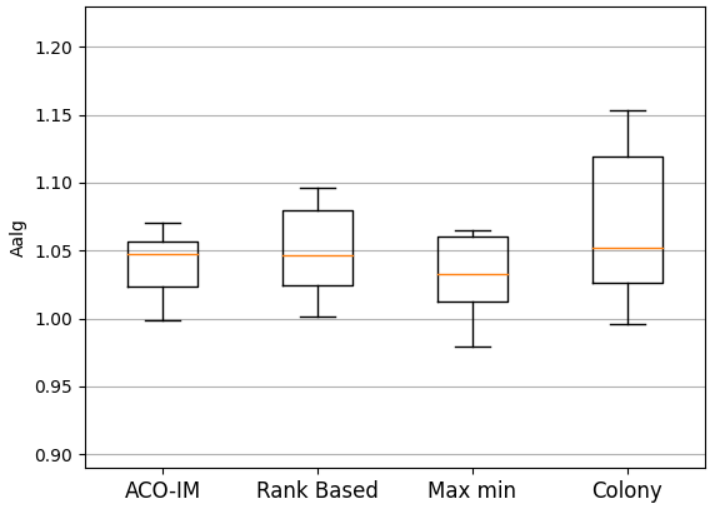

Figura 1: Medida A_{alg} de los algoritmos bajo el modelo IC en grafos reales

mejores resultados puede llegar a conseguir, con un A_{alg} máximo de 1.15 es decir, hasta un 15 % mejor que el básico.

En la figura 2 podemos ver otra serie de observaciones. En este caso, es el algoritmo ACO-IM el que en algunas ocasiones llega a tener peor rendimiento que el original, mientras el resto de algoritmos superan al básico en todo grafo. Puede verse como en esta ocasión, es el modelo Rank-Based el que menor variabilidad tiene, seguido por el Max-Min. En cuanto a rendimiento, en esta ocasión también consigue los mejores resultados el Colony, con A_{alg} de algo menos de 1.15, aunque con tanta variabilidad, es posible que en este caso sea mejor quedarnos con el Rank-Based ya que garantiza un mejor caso promedio (aunque su mediana sea algo menor).

En la figura 3 podemos volver a notar como tanto el ACO-IM como el Max-Min tienen algunos casos de peor rendimiento que el algoritmo básico. Todos se encuentran bastante parecidos en tema de variabilidad, aunque como de costumbre, el Colony sigue siendo el que mayor tiene de ésta. En este caso, si hubiese que quedarse con alguno, deberíamos decantarnos por el Colony que llega a alcanzar un A_{alg} de algo más de 1.2, aunque sí que es cierto que en esta ocasión la mediana del Rank-Based es algo más alta y sería debatible cuál lo está haciendo mejor.

Hemos visto como en los tres modelos, en el caso de los grafos reales, son los algoritmos de Rank-Based y de Colony los que mejor lo hacen, mientras que el ACO-IM no suele destacar demasiado y Max-Min directamente tiene un rendimiento general bastante pésimo, incluso peor que el básico.

5.8. Resultados obtenidos en grafos Erdős-Rényi

Ahora continuaremos a analizar estos mismos algoritmos con los grafos artificiales de Erdős-Rényi.

Volveremos a representar en otra tabla todos los resultados obtenidos sobre la media de los A_{alg} y sus respectivos rangos en los grafos Erdős-Rényi por cada modelo de difusión distinto.

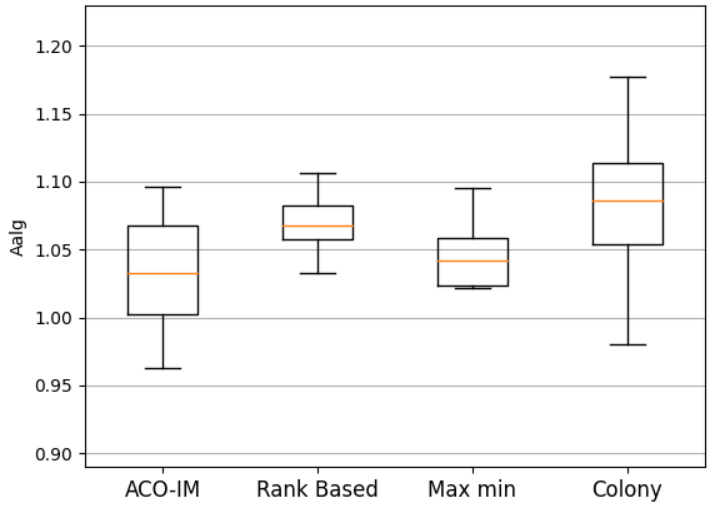

Figura 2: Medida A_{alg} de los algoritmos bajo el modelo WC en grafos reales

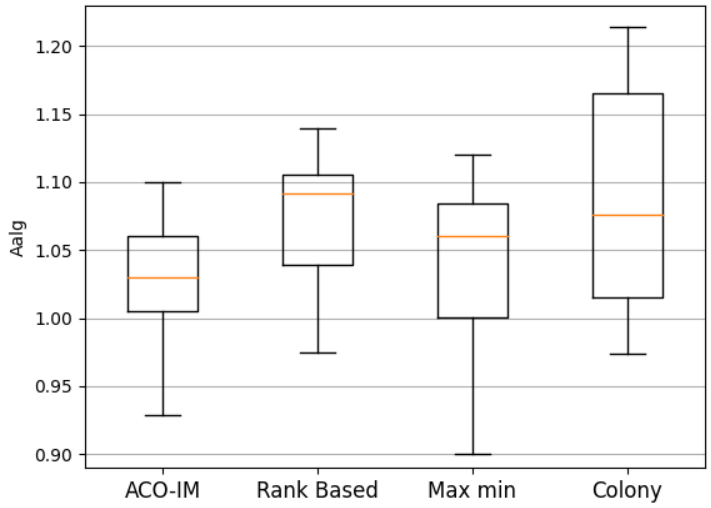

Figura 3: Medida A_{alg} de los algoritmos bajo el modelo LT en grafos reales

	IC		WC		LT	
	$\overline{A_{alg}}$	Rango	$\overline{A_{alg}}$	Rango	$\overline{A_{alg}}$	Rango
Basico	1369.45	2308.49	1352.52	2328.30	3129.1	9383.0
ACO-IM	1369.28	2327.47	1349.90	2400.97	3166.45	9752.0
Rank-Based	1370.45	2326.48	1356.46	2353.01	3189.87	9683.0
Max-Min	1336.03	2344.55	1304.02	2336.93	3033.27	9666.0
Colony	1367.72	2315.39	1351.58	2339.34	3181.25	9682.0

Tabla 3: A_{Alg} conseguido en grafos Erdős-Rényi

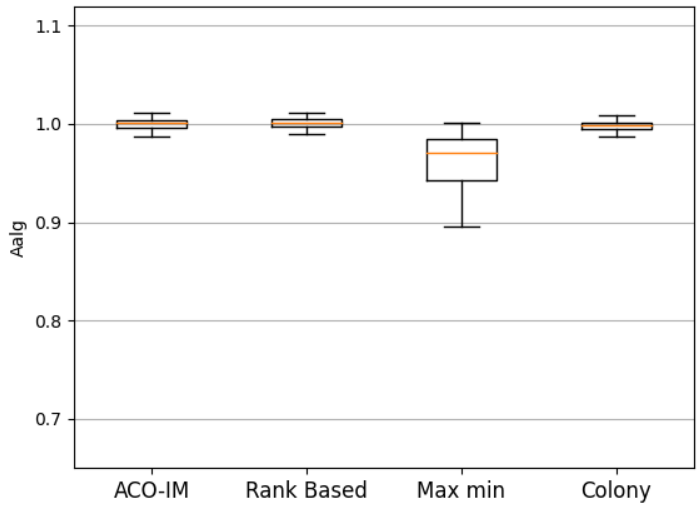

Figura 4: Medida A_{alg} de los algoritmos bajo el modelo IC en grafos Erdős-Rényi

Podemos apreciar en la tabla 3 ocurrencias similares a las expuestas en la tabla 2, pero con sutiles diferencias que haremos notar. Sí que es cierto que el modelo IC sigue rindiendo peor que el modelo LT, pero en esta ocasión podemos apreciar como el modelo IC tiene mejor A_{alg} de media respecto al modelo WC. Para los grafos Erdős-Rényi, en vez de partir de unos valores de umbral de percolación previamente calculados, podemos computarlos por nuestra cuenta con la fórmula $p^* = \frac{1}{D(G)}$. Esto quiere decir que en el caso de que todos los nodos tengan un grado parecido, al aplicar el modelo IC tendrán un peso de arista similar al que tendrían en el caso del modelo WC, que es lo que está pasando precisamente aquí. Para poder conseguir grafos Erdős-Rényi con una sola componente conexa, necesitamos que haya una gran probabilidad de que se formen aristas, por lo que en general el grafo será bastante conexo. Al usar el modelo WC tendremos un peso de arista bastante bajo por este motivo, casi igual al umbral de percolación al del modelo IC, por lo que ambos tendrán una eficacia similar.

Al igual que en el caso de los grafos reales, también fijaremos los límites del eje Y en las gráficas para poderlas comparar con mayor facilidad (A un mínimo de 0.65 y un máximo de 1.12)

En la figura 4 podemos ver unos resultados mucho más dispares a los que hemos visto con anterioridad. Podemos argumentar con claridad que son bastante peores, ya que todo y cada uno de los algoritmos es capaz de hacerlo peor en algunos grafos que el algoritmo básico. No solo eso, sino que la mejora obtenida en el mejor de los casos respecto al algoritmo básico es consiguiendo un A_{alg} algo menor de 1.02, osease, ni siquiera un 2 % mejor. Además, el algoritmo Max-Min es bastante remarcable en esta ocasión, ya que no es capaz de conseguir un $A_{alg} > 1$ en ningún caso, es decir, lo está haciendo directamente peor que el algoritmo básico en todos los grafos, con un mínimo de A_{alg} sobre el 0.9. En el caso del resto del algoritmos no es mucho mejor, los otros 3 tienen una variabilidad bastante baja y se mantienen con la mediana sobre el 1, consiguiéndolo hacer a veces peor y a veces mejor, pero en promedio igual que el original.

En la figura 5 nos encontramos resultados aparentemente similares, pero bajo mayor escrutinio nos damos cuenta de que realmente son peores. Sigue pasando el mismo fenómeno

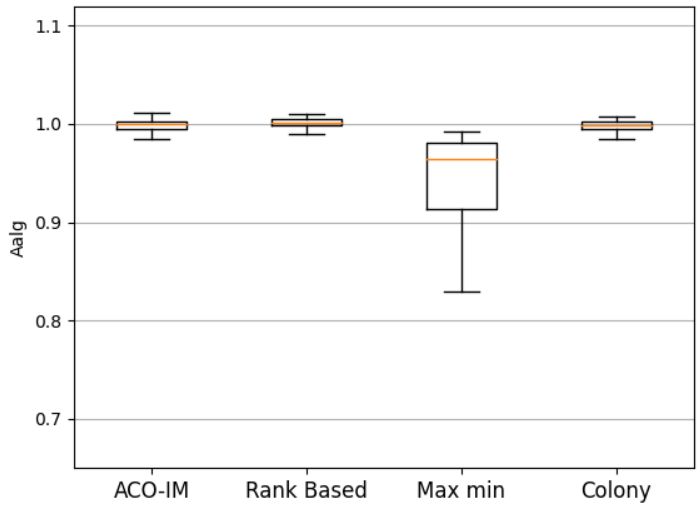

Figura 5: Medida A_{alg} de los algoritmos bajo el modelo WC en grafos Erdős-Rényi

de que todo algoritmo lo haga peor que el algoritmo básico en algunas instancias de los grafos, y a su vez, el algoritmo Max-Min sigue destacando por su falta rendimiento, está vez ni siquiera llegando su máximo A_{alg} hasta 1, y desplomando su mínimo A_{alg} hasta casi 0.825. En el caso del resto de algoritmos tampoco podemos decir que lo hayan hecho demasiado peor, simplemente su variabilidad ha disminuido un poco más, haciendo que en los casos que lo hacían mejor lo hagan algo peor, pero también mejoren sus peores casos, pero aún siguen teniendo el mismo problema que veíamos en el modelo de difusión anterior.

Por último, en la figura 6 podemos apreciar algunos cambios, aunque seguimos viendo los mismos patrones que en casos anteriores. Todos los algoritmos siguen teniendo en algunos casos peores rendimientos respecto al algoritmo básico, aunque esta vez al menos no hay ninguno que lo haga directamente peor y el Max-Min se obtiene algún resultado puntual mejor, aunque por lo general sigue haciéndolo mal (su tercer percentil ya se encuentra por debajo de $A_{alg} = 0$). También podemos apreciar como además el Max-Min ha conseguido batir su récord en cuanto a valor mínimo, llegando a un A_{alg} menor que 0.7. En cuanto al resto de algoritmos, su variabilidad ha aumentado con creces, pero al seguir su mediana en $A_{alg} = 1$, implica que sus mejores resultados han mejorado hasta $A_{alg} = 1{,}1$, pero también sus peores resultados han empeorado, hasta un A_{alg} de 0.9

Podemos comprobar que en el caso de estos grafos artificiales de Erdős-Rényi, conseguimos resultados de propagación de la influencia bajos. Esto puede ir ligado al comportamiento de estos grafos, donde no se forman demasiados clusters y entonces no podemos sacar partido de implementar mejoras, ya que los resultados no varían demasiado. Como mencionamos previamente, es inevitable evitar que los nodos estén tan interconectados entre sí, o no habríamos obtenido una sola componente conexa, imprescindible para poder realizar el estudio de la maximización de la influencia en estos grafos. También puede ir ligado a que a la hora de establecer cuantos nodos forman parte de nuestro conjunto de nodos semilla $\mathcal{S}$ al elegir las distintas k en la formula 15, es posible que sea un número demasiado pequeño para el tamaño de los grafos. Si partimos de una k mínima de 0.01, quiere decir que en el más pequeño de nuestros grafos, solo formarán parte de los nodos semilla el 1 % de los nodos, es decir, un sólo nodo. Al final estaremos consiguiendo pro-

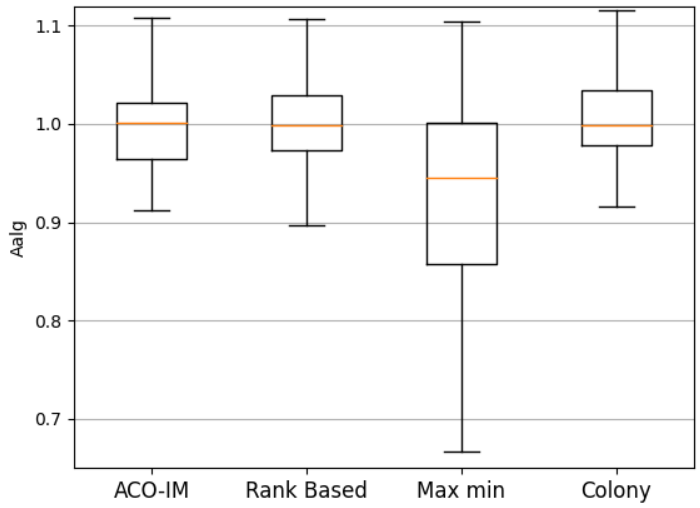

Figura 6: Medida A_{alg} de los algoritmos bajo el modelo LT en grafos Erdős-Rényi

	IC		WC		LT	
	$\overline{T_{alg}}$	Rango	$\overline{T_{alg}}$	Rango	$\overline{T_{alg}}$	Rango
Basico	34.42	86.86	34.64	87.23	34.91	93.53
ACO-IM	29.19	76.22	29.41	76.12	29.53	83.16
Rank-Based	38.56	94.61	38.45	96.35	38.90	103.93
Max-Min	32.30	84.76	32.68	87.62	32.78	92.22
Colony	35.58	95.22	35.48	95.39	35.81	101.44

Tabla 4: Tiempos obtenidos en los grafos Erdős-Rényi

pagaciones de influencia $\sigma(\mathcal{S})$ ínfimas, ya que la probabilidad de que tenga efecto es casi nula. Obtener mejores soluciones no va a hacer que nuestra propagación de la influencia $\sigma(\mathcal{S})$ sea mucho más masiva, y por lo tanto el A_{alg} no va a variar demasiado.

5.9. Tiempos obtenidos en los grafos Erdős-Rényi

Ahora pasaremos a hablar de los tiempos que ha tardado cada algoritmo en cada modelo, al ejecutar el algoritmo y posteriormente la simulación de propagación de la influencia $\sigma(\mathcal{S})$. Únicamente se han tomado los tiempos en las redes artificiales Erdős-Rényi, pero al depender la complejidad computacional de el numero de nodos $V(G)$ y el número de hormigas $|\mathcal{H}|$, es igual que tomemos los tiempos de grafos reales que de generados artificialmente. La medida que se está representando no es directamente el tiempo, sino que al igual que con los A_{alg} anteriores, se trata del radio entre el tiempo que tardo el algoritmo que estemos comparando con el algoritmo básico, variando el tamaño k de los nodos semilla $\mathcal{S}$, al igual que hacíamos con las formulas 15 y 16. En esta ocasión, un valor alto del radio implica que ha tardado más que el algoritmo base, y por lo tanto, va en su detrimento.

Como en ocasiones anteriores, primero mostraremos una tabla con las medias del tiempo y sus respectivos rangos en cada uno de los modelos de difusión:

Podemos ver en la tabla 4 que no hay gran diferencia entre los tiempos que tarda un algoritmo en concreto entre los distintos modelos de difusión usados. Sí que hay diferencias

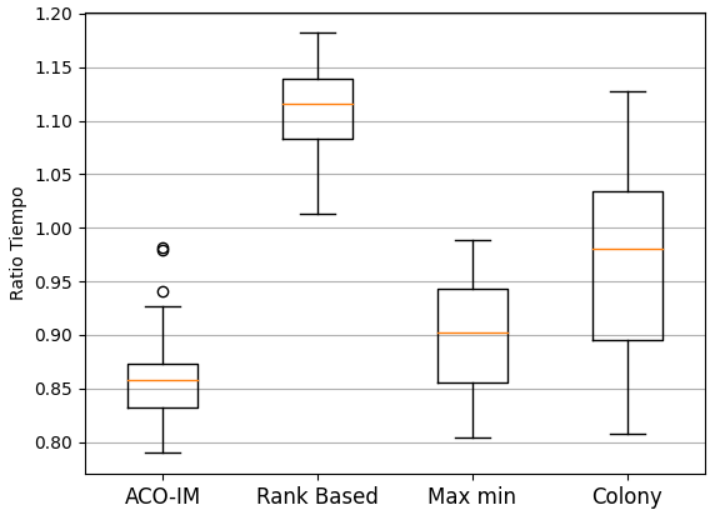

Figura 7: Tiempo tardado de los algoritmos bajo el modelo IC en grafos Erdős-Rényi

dependiendo del algoritmo que se use, por lo que lo veremos más en profundidad en las siguientes gráficas.

Puede apreciarse que en estás tres gráficas no se han eliminado los valores atípicos, ya que no entorpecían la visualización de éstas.

En la figura 7 podemos apreciar como en está ocasión es el algoritmo ACO-IM el que lo hace particularmente bien, siendo junto al Max-Min los que siempre tardan menos que el algoritmo base. Mientras tanto, el algoritmo Colony no destaca demasiado, en ocasiones lo hace mejor y en otras peor, y por último el Rank-Based es el más costoso en cuanto a tiempo de todos, siempre tardando más que el algoritmo base, hasta algo más de un 10 % más.

En la figura 8 podemos ver un patrón similar, por no decir idéntico, donde los algoritmos ACO-IM y Max-Min siguen destacando, el Colony vuelve a no hacerlo ni particularmente bien ni mal, y el Rank-Based continua en su linea de tardar más que el resto. Esto es acorde con lo desarrollado, ya que las diferencias de tiempos entre los distintos modelos deben surgir en la simulación de la propagación de la influencia $\sigma(\mathcal{S})$, al ser los algoritmos iguales para todos los distintos modelos de difusión. Como se propuso anteriormente, la propagación de la influencia $\sigma(\mathcal{S})$ del modelo IC y el modelo WC siguen el mismo patrón, solo diferenciando los pesos de las aristas, por lo que es lógico que tarden tiempos similares.

En la figura 9 vemos alguna discrepancia respecto a las dos anteriores, donde ahora se puede apreciar como ninguno de los algoritmos lo hace siempre más rápido o más lento que el básico. El algoritmo Max-Min sigue siendo el que mejores tiempos consigue, seguido por el Colony, que casi empata con el ACO-IM y por último sigue el Rank-Based, aunque en está ocasión no es tan fatídico como en el pasado. El motivo de estos tiempos más igualados debe de estar en las simulaciones, donde se tarda más tiempo que de costumbre, y como en éstas los tiempos son similares, acaba suavizando la gráfica.

En general, los algoritmos que mejor lo han hecho en cuanto a tiempo son los ACO-IM y Max-Min, que coinciden con los algoritmos que peores resultados obtenían en las medidas de A_{alg}, especialmente el Max-Min. En cambio, el Colony obtiene tiempos por lo general

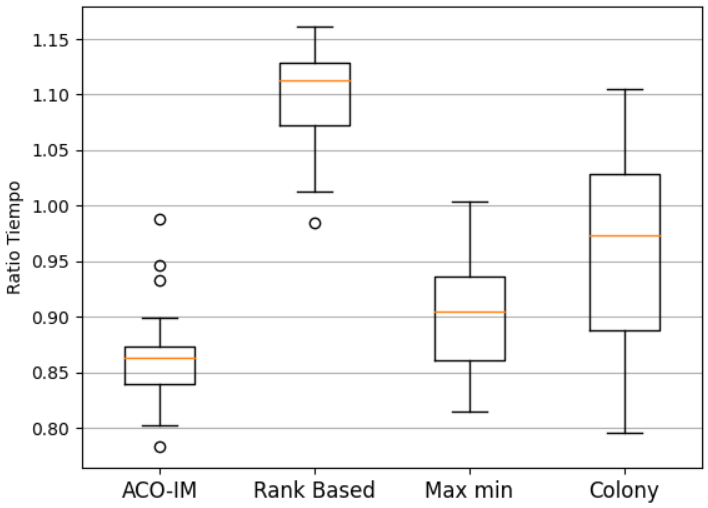

Figura 8: Tiempo tardado de de los algoritmos bajo el modelo WC en grafos Erdős-Rényi

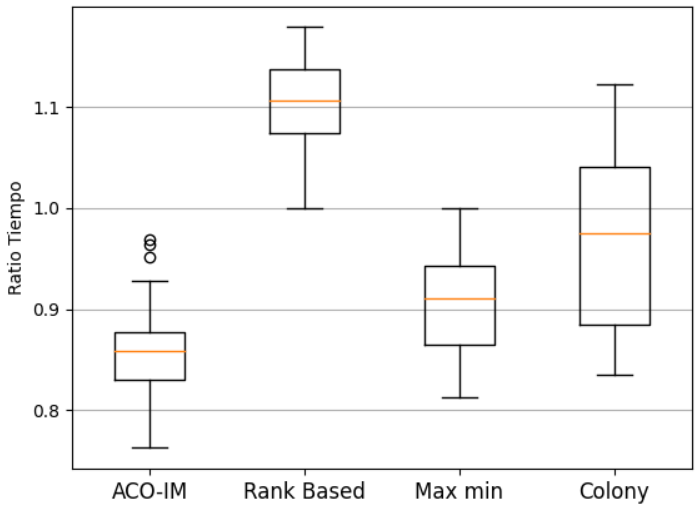

Figura 9: Tiempo tardado de de los algoritmos bajo el modelo LT en grafos Erdős-Rényi

promedios, y en retorno tiene también valores de A_{alg} bastante positivos, por lo que es buen candidato si tuviésemos que quedarnos con una única variante de la metaheurística de colonia de hormigas.

6. Conclusión y trabajos futuros

Vamos a concluir este trabajo sacando las conclusiones y lecciones aprendidas, además de qué posibles enfoques se le podría dar al trabajo en el futuro.

Con este TFG se han conseguido alcanzar los objetivos propuestos. Para empezar, hemos conseguido implementar de forma satisfactoria el algoritmo ACO-IM propuesto por Singh et al. 2020, únicamente partiendo del pseudocódigo administrado. Asimismo, también se ha conseguido implementar los cambios de las feromonas (Pérez-Carabaza et al. 2022), realizando los ajustes para que cada una de las 3 variantes vistas puedan funcionar. Posteriormente se ha realizado toda la batería de pruebas, y hemos analizado los resultados obtenidos y comparado el desempeño de cada uno de los algoritmos implementados. Por último, se ha demostrado la hipótesis, probando que el algoritmo ACO-IM que están usando los autores no está totalmente optimizado y hay espacio para mejorar ciertos aspectos y poder hallar mejores soluciones.

Es cierto que se intentó conseguir los resultados mencionados en el articulo de los autores (Singh et al. 2020) de ACO-IM, pero los resultados no llegaron a ser tan buenos como los mostrados en el artículo; no se conseguía la propagación de la influencia $\sigma(\mathcal{S})$ mostrada con los cinco grafos del artículo. Al no tener ni el código del artículo ni todos los grafos que ellos usaban de prueba, no fue posible determinar exactamente cuál es la causa de este comportamiento inesperado.

Luego enfocamos los esfuerzos en tratar de conseguir mejores resultados recurriendo a otros métodos, como todas las variantes vistas (Pérez-Carabaza et al. 2022), y aunque tampoco se llegó a los resultados mostrados en ACO-IM Singh et al. 2020, sí que se notaron mejoras en el caso de algunas variantes, como Rank-Based.

Puede verse como logrado el objetivo de analizar distintas variantes del algoritmo original, ya que se han implementado distintas variantes consiguiendo ver alguna diferencia entre ellos. Cabe decir que hay ciertas características del ACO-IM que solapaban los efectos que tenia la variación en la actualización de feromona, como el mecanismo contra la convergencia prematura, y tuvieron que ser eliminados en todas estas variantes para poder conseguir resultados más fiables.

Por otro lado, el uso de grafos artificiales de Erdős-Rényi consiguió mostrar la optimización temporal de los algoritmos, al poderlos ejecutar en grafos más pequeños, en comparación a los grafos reales, cuyo gran tamaño dificultaba la toma de medidas. No obstante, es notable que el uso de estos grafos no es adecuado para el problema de la maximización de la influencia y por lo tanto no muestran la eficiencia de unos algoritmos frente a otros.

En resumen, se considera que se partió de la hipótesis correcta, de que un cambio en la actualización de las feromonas tendría un impacto en el resultado del algoritmo, y los resultados obtenidos en este trabajo acompañan, mostrando que tanto las variantes Rank-Based como Colony podrían ser candidatas a ser usadas en lugar de ACO-IM.

6.1. Trabajos futuros

Concluimos con las distintas vías que podemos tomar a la hora de retomar el trabajo en el futuro:

- Utilizar un mayor número de grafos reales para las pruebas del trabajo, preferiblemente encontrando alguno de menor tamaño para un menor coste computacional.
- Buscar otra forma de variar el fitness con el objetivo de encontrar soluciones más interesantes.
- Usar otro generadores de grafos que se ajusten a las necesidades del problema de la maximización de la influencia.

Referencias

Aghaee, Zahra et al. (nov. de 2021). «A survey on meta-heuristic algorithms for the influence maximization problem in the social networks». En: *Computing* 103.11, págs. 2437-2477. ISSN: 1436-5057. DOI: 10.1007/s00607-021-00945-7. URL: https://doi.org/10.1007/s00607-021-00945-7.

Banerjee, Suman, Mamata Jenamani y Dilip Kumar Pratihar (sep. de 2020). «A survey on influence maximization in a social network». En: *Knowledge and Information Systems* 62.9, págs. 3417-3455. ISSN: 0219-3116. DOI: 10.1007/s10115-020-01461-4. URL: https://doi.org/10.1007/s10115-020-01461-4.

Bullnheimer, Bernd, Richard Hartl y Christine Strauss (ene. de 1999). «A New Rank Based Version of the Ant System - A Computational Study». En: *Central European Journal of Operations Research* 7, págs. 25-38.

Chen, Wei, Yajun Wang y Siyu Yang (2009). «Efficient Influence Maximization in Social Networks». En: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '09. Paris, France: Association for Computing Machinery, págs. 199-208. ISBN: 9781605584959. DOI: 10.1145/1557019.1557047. URL: https://doi.org/10.1145/1557019.1557047.

Domingos, Pedro y Matt Richardson (2001). «Mining the Network Value of Customers». En: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '01. San Francisco, California: Association for Computing Machinery, págs. 57-66. ISBN: 158113391X. DOI: 10.1145/502512.502525. URL: https://doi.org/10.1145/502512.502525.

Dorigo, M. y L.M. Gambardella (1997). «Ant colony system: a cooperative learning approach to the traveling salesman problem». En: *IEEE Transactions on Evolutionary Computation* 1.1, págs. 53-66. DOI: 10.1109/4235.585892.

Erdös, P y A Rényi (1959). «On Random Graphs I». En: *Publicationes Mathematicae Debrecen* 6, págs. 290-297.

Erkol, Şirag, Claudio Castellano y Filippo Radicchi (oct. de 2019). «Systematic comparison between methods for the detection of influential spreaders in complex networks». En: *Scientific Reports* 9.1, pág. 15095. ISSN: 2045-2322. DOI: 10.1038/s41598-019-51209-6. URL: https://doi.org/10.1038/s41598-019-51209-6.

Kempe, David, Jon Kleinberg y Éva Tardos (2003). «Maximizing the Spread of Influence through a Social Network». En: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '03. Washington, D.C.: Association for Computing Machinery, págs. 137-146. ISBN: 1581137370. DOI: 10.1145/956750.956769. URL: https://doi.org/10.1145/956750.956769.

Kumar, D Nagesh y M Janga Reddy (oct. de 2006). «Ant Colony Optimization for Multi-Purpose Reservoir Operation». En: *Water Resources Management* 20. DOI: 10.1007/s11269-005-9012-0.

Patwardhan, Siddharth, Filippo Radicchi y Santo Fortunato (2022). *Influence Maximization: Divide and Conquer*. arXiv: 2210.01203 [physics.soc-ph].

Pérez-Carabaza, Sara, Akemi Gálvez y Andrés Iglesias (2022). «Rank-Based Ant System with Originality Reinforcement and Pheromone Smoothing». En: *Applied Sciences* 12.21. ISSN: 2076-3417. DOI: 10.3390/app122111219. URL: https://www.mdpi.com/2076-3417/12/21/11219.

Singh, Shashank Sheshar et al. (jul. de 2020). «ACO-IM: maximizing influence in social networks using ant colony optimization». En: *Soft Computing* 24.13, págs. 10181-10203. ISSN: 1433-7479. DOI: 10.1007/s00500-019-04533-y. URL: https://doi.org/10.1007/s00500-019-04533-y.

Stützle, Thomas y Holger H. Hoos (2000). «MAX–MIN Ant System». En: *Future Generation Computer Systems* 16.8, págs. 889-914. ISSN: 0167-739X. DOI: https://doi.org/10.1016/S0167-739X(00)00043-1. URL: https://www.sciencedirect.com/science/article/pii/S0167739X00000431.

Anexo I: Nomenclatura

Se recoge a continuación un cuadro con la nomenclatura que se ha utilizado para describir este trabajo. Consiste en una aglomeración de términos recogidos de otros autores nombrados en la bibliografía, adaptados para que no haya ningún conflicto entre sí:

Símbolo	Significado
$G(V, E, W)$	Grafo dirigido y con pesos
$V(G)$	Conjunto de nodos del grafo
$E(G)$	Conjunto de aristas del grafo
$W(G)$	Conjunto de pesos de probabilidad de difusión del grafo
$\theta(G)$	Conjunto de pesos de umbral de influencia del grafo
$w(u, v)$	Peso de la arista del nodo u al nodo v
$\mathcal{N}^{out}(u)$	Vecindad de salida del nodo u
$\mathcal{N}^{in}(u)$	Vecindad de entrada del nodo u
$D^{out}(u)$	Grado de salida del nodo u
$D^{in}(u)$	Grado de entrada del nodo u
k	Tamaño de la solución
$\mathcal{H}$	Conjunto de agentes hormigas
h_z	Hormiga z perteneciente al conjunto de hormigas $\mathcal{H}$
$\mathcal{S}$	Conjunto de nodos semilla
$s(h_z)$	Conjunto de nodos que conforman la solución de la hormiga h_z
$f(s(h_z))$	Fitness de la solución generada por la hormiga A_i
$\tau_0(u)$	Feromona del nodo u en la iteración $i = 0$
$\sigma(\mathcal{S})$	propagación de la influencia conseguida por el conjunto de nodos semilla $\mathcal{S}$

Tabla 5: Nomenclatura usada

Anexo II: Grafos

A continuación se encuentran todos los grafos reales usados, con su número de nodos, aristas y valor de percolación:

Nombre	Nodos	Aristas	Valor de percolación
email.txt	1133	5451	0.056
out.maayan-faa	1226	2408	0.163
out.petster-friendships-hamster-uniq	1788	12476	0.025
out.opsahl-ucsocial	1893	13835	0.023
Yeast.paj	2224	6609	0.071
out.moreno_health_health	2539	10455	0.117
socfb-USFCA72.mtx	2672	65244	0.011
japanesebookinter_st.txt	2698	7995	0.030
out.opsahl-openflights	2905	15645	0.020
socfb-Pepperdine86.mtx	3440	152003	0.007
socfb-Wesleyan43.mtx	3591	138034	0.009
socfb-Mich67.mtx	3745	81901	0.011
socfb-Bucknell39.mtx	3824	158863	0.008
socfb-Howard90.mtx	4047	204850	0.006
CA-GrQc.txt	4158	13422	0.091
out.reactome	5973	145778	0.011
out.subelj_jung-j_jung-j	6120	50290	0.009
p2p-Gnutella08.txt	6299	20776	0.046
out.subelj_jdk_jdk	6434	53658	0.009
socfb-UChicago30.mtx	6561	208088	0.008
socfb-UC64.mtx	6810	155320	0.010

Tabla 6: Grafos usados

A su vez mostraremos los grafos Erdős-Rényi reales usados, con su número de nodos, aristas y valor de percolación. El nombre de los grafos va en función de el número de nodos y la probabilidad (multiplicada por 100) usadas en su creación:

Nombre	Nodos	Aristas	Valor de percolación
N100P10.gml	100	948	0.10548523206751054
N100P4.gml	100	462	0.21645021645021645
N100P6.gml	100	590	0.1694915254237288
N100P8.gml	100	826	0.12106537530266344
N200P10.gml	200	4036	0.049554013875123884
N200P4.gml	200	1550	0.12903225806451613
N200P6.gml	200	2372	0.08431703204047218
N200P8.gml	200	3054	0.06548788474132286
N300P10.gml	300	9146	0.03280122457905095
N300P4.gml	300	3602	0.08328706274292061
N300P6.gml	300	5470	0.054844606946983544
N300P8.gml	300	7198	0.04167824395665463
N400P10.gml	400	15816	0.025290844714213456
N400P4.gml	400	6408	0.062421972534332085
N400P6.gml	400	9448	0.04233700254022015
N400P8.gml	400	12850	0.0311284046692607
N500P10.gml	500	24928	0.020057766367137356
N500P4.gml	500	9976	0.05012028869286286
N500P6.gml	500	15146	0.03301201637396012
N500P8.gml	500	19954	0.025057632554876214
N600P10.gml	600	35802	0.01675884028825205
N600P4.gml	600	14032	0.042759407069555305
N600P6.gml	600	21874	0.02742982536344519
N600P8.gml	600	29046	0.020656889072505683
N700P10.gml	700	48896	0.014316099476439791
N700P4.gml	700	19564	0.03578000408914332
N700P6.gml	700	29398	0.02381114361521192
N700P8.gml	700	39326	0.0177999288002848
N800P10.gml	800	63658	0.012567155738477487
N800P4.gml	800	25078	0.031900470531940346
N800P6.gml	800	38444	0.02080948912704193
N800P8.gml	800	51086	0.015659867674118155
N900P10.gml	900	81464	0.011047824806049298
N900P4.gml	900	32178	0.027969420100689913
N900P6.gml	900	48284	0.018639715019468147
N900P8.gml	900	64148	0.014030055496663965
N1000P10.gml	1000	99772	0.01002285210279437
N1000P4.gml	1000	39802	0.025124365609768353
N1000P6.gml	1000	59854	0.01670732114812711
N1000P8.gml	1000	79978	0.012503438445572534

Tabla 7: Grafos Erdős-Rényi usados