Facultad de Ciencias

Búsqueda de comunidades con valores atípicos mediante el uso de algoritmos genéticos

(Community search with outliers using genetic algorithms)

Trabajo de Fin de Máster
para acceder al

MÁSTER EN MATEMÁTICAS Y COMPUTACIÓN

Autor: Guzmán Rodríguez Fernández

Director\es: Camilo Palazuelos Calderón

Resumen

En este trabajo, exploraremos una variante de la búsqueda de comunidades en grafos, un tema ampliamente estudiado. Esta variante se centra en lo siguiente: dentro de una red, contamos con una serie de elementos de interés que están relacionados entre sí, y nuestro objetivo es encontrar elementos similares a estos nodos de interés. Nuestra meta es identificar una subred que incluya no solo los elementos de interés, sino también otros nodos similares, lo que nos permitirá obtener una visión más completa y detallada de esta comunidad. Sin embargo, surge un desafío: si uno de los elementos no está verdaderamente relacionado con el resto, la comunidad resultante puede no ser del todo precisa o, al menos, no sería tan precisa como si excluyéramos este elemento atípico (outlier). Por lo tanto, abordaremos este problema considerando la posibilidad de eliminar hasta k elementos al construir la nueva red.

Palabras clave: grafo, outlier, NP-duro, algoritmo genético, comunidad

Abstract

In this paper, we will explore a variant of community search in networks, a widely studied topic. This variant focuses on the following: within a network, we have a number of elements of interest that are related to each other, and our goal is to find elements similar to these nodes of interest. Our goal is to identify a sub-network that includes not only the elements of interest, but also other similar nodes, which will allow us to obtain a more complete and detailed view of this community. However, a challenge arises: if one of the elements is not truly related to the rest, the resulting community may not be entirely accurate, or at least not as accurate as if we exclude this outlier. Therefore, we will address this problem by considering the possibility of eliminating up to k elements when constructing the new network.

Key words: graph, outlier, NP-hard, genetic algorithm, community

Índice

1. Introducción

1.1. Motivación

En la actualidad, las redes sociales poseen un gran poder en la forma en que las personas se comunican, interactúan y obtienen información. Con millones de usuarios en todo el mundo, plataformas como Facebook, Instagram y Twitter se han convertido en herramientas indispensables para compartir noticias, opiniones, ideas y experiencias. En consecuencia, se ha creado un enorme conjunto de comunidades, ya sea un grupo de amigos, familia, gente que apoya a un mismo equipo de fútbol, entre otros.

Detectar estas comunidades es algo que puede resultar de interés. Por ejemplo, si contamos con una serie de perfiles problemáticos relacionados, nos gustaría conocer y vigilar a la gente de su entorno. Esta es una de las muchas utilidades que tiene la búsqueda de comunidades.

Ahora bien, ¿y si algunas de esas personas que hemos considerado problemáticas no está realmente relacionada con el resto? En ese caso, buscar una comunidad que contenga a todos puede ser difícil, y los resultados obtenidos pueden no ajustarse a una buena solución. Para resolver esto, vamos a introducir una nueva variable al problema, que es poder prescindir de algunos elementos a la hora de buscar la comunidad a la que pertenecen los perfiles objetivo.

Este trabajo va a ser una ampliación de [1], donde aplican una serie de algoritmos que aproximan la solución óptima. En nuestro caso, vamos a aplicar algoritmos genéticos para resolver estos problemas, comparando los resultados tanto en tiempo de ejecución como en precisión de la solución.

1.2. Definición del problema

En primer lugar, vamos a definir formalmente el problema original, la modificación que lleva al problema que abordamos, las medidas de cohesión utilizadas y los tres enfoques que se aplican en [1].

Dado un grafo $G = (V, E)$, que representa la red original, y un subconjunto de vértices $Q \subseteq V$, que son los vértices objetivo, encontrar un subgrafo H de G que contenga todos los vértices de Q.

Sin embargo, como se ha explicado anteriormente, añadimos una circunstancia adicional, que es que podemos prescindir de un número k de vértices de Q para generar el subgrafo H.

Las medidas que vamos a utilizar como funciones de cohesión son el diámetro del subgrafo generado H [3] y el mínimo grado [8].

- $diam(H) = \text{máx}\{d_H(u, v) : u, v \in V_H\}$

- $d_H(u, v)$ es el camino más corto entre los vértices u y v en el grafo H
- $min\_deg(H) = \text{mín}\{deg(u) : u \in V_H\}$, donde $deg(u)$ se refiere al grado de u

La idea que proponen en el artículo es optimizar una de las medidas utilizando la otra como restricción. Hay que tener en cuenta que en todos los casos las únicas entradas son: $G = (V, E)$, $Q \subseteq V, k$. Siendo k el número máximo de vértices de Q que podemos descartar.

El problema de utilizar el diámetro como medida es su complejidad computacional. La complejidad de calcular el diámetro de un grafo es $\mathcal{O}(n^3)$, siendo n el número de vértices del grafo, algo que es poco asumible, dado que se implementa un algoritmo heurístico, y esto significaría que tendría que hacerse el cálculo un elevado número de veces.

La solución a este problema es utilizar algoritmos de aproximación como los que se explican en [2]. De entre todos los descritos hemos decidido estudiar los dos siguientes.

El primero de ellos aplica la búsqueda en anchura (BFS) a un vértice aleatorio r, que tiene una complejidad de $\mathcal{O}(n + m)$, siendo n el número de vértices y m el número de aristas. Al aplicar este algoritmo se obtiene un árbol en la que la raíz es en vértice r y la distancia a cada nodo es la mínima distancia a dicho nodo en el grafo original. Con una sola búsqueda se obtiene un factor de aproximación 2.

Algorithm 1 Aproximación del diámetro de un grafo con factor 2

Input: $G = (V, E)$
Output: $diam$
$r \leftarrow v \in V$
$d = \text{máx}_{v \in V}\, \text{dist}(r, v)$
return $2 \cdot d$

Demostración: Si tomamos el diámetro D' como el doble de la distancia al vértice más alejado v, el diámetro real D, que sería $d(u, w)$ siendo u y w los vértices mas alejados entre sí, siempre va a ser menor o igual a D', ya que, en el peor caso, sería $d(u, r) + d(r, w)$, siendo $d(u, r)$ y $d(r, w) \leq d(r, v)$.

Por tanto, este algoritmo tiene una aproximación de factor 2, es decir, la solución encontrada por el algoritmo se encuentra a una distancia como máximo dos veces mayor que la solución óptima.

Esta aproximación del diámetro, si bien es muy eficiente, tiene un factor muy alto. Por ello, se va a proponer el uso de otro algoritmo de aproximación para el cálculo del diámetro. Dicho algoritmo, descrito en [6] tiene un factor de aproximación de 1,5. La idea de este algoritmo es obtener un conjunto de vértices aleatorios S de tamaño $\sqrt{n}$. Después, se busca el vértice w más alejado de todos los vértices del conjunto S, siendo $D1$ la distancia a ese vértice desde el nodo de S más alejado. Desde ese vértice, se obtienen los $\sqrt{n}$ vértices más cercanos, y se repite el mismo

proceso, buscar el vértice x más alejado de estos y la mayor distancia hasta él, $D2$. Finalmente, el diámetro obtenido es: $diam = \frac{3}{2} \cdot \text{máx}(D1, D2)$.

Algorithm 2 Aproximación del diámetro de un grafo con factor 1.5

Input: $G = (V, E)$
Output: $diam$
$S \subseteq V, |S| = \sqrt{|V|}$
$w \leftarrow \arg\text{máx}_{v \in V}\ \text{mín}_{s \in S}\ \text{dist}(v, s)$
$D1 \leftarrow \text{mín}_{s \in S}\ \text{dist}(w, s)$
$S2 = \{v_i \in V \mid \text{dist}(v, v_i)$ es uno de los $\sqrt{|V|}$ valores más pequeños$\}$
$x \leftarrow \arg\text{máx}_{v \in V}\ \text{mín}_{s \in S2}\ \text{dist}(v, s)$
$D2 \leftarrow \text{mín}_{s \in S2}\ \text{dist}(x, s)$
return $\frac{3}{2} \cdot \text{máx}(D1, D2)$

Este otro algoritmo, a pesar de tener una alta probabilidad de obtener un diámetro con el factor adecuado, tiene una pequeña probabilidad de que el resultado se salga de la aproximación marcada. A pesar de ello, es un fallo que se puede asumir en un algoritmo heurístico como este, teniendo en cuenta que la complejidad temporal debe ser limitada. El coste de este método es $\mathcal{O}(m\sqrt{n \log(n)})$.

En este trabajo no se implementa este segundo algoritmo dado que el aumento del coste implica un gran aumento en el tiempo de ejecución de toda la experimentación. Por tanto, queda como trabajo futuro realizar una comparación entre el uso de ambos algoritmos.

1.3. Problemas presentados

Vamos a ver los tres problemas expuestos en el artículo.

Problema 1 (ORIGINAL-P1): Este problema se basa en minimizar el diámetro utilizando el mínimo grado (δ_{min}) como restricción. Dado que el número de vértices que podemos descartar no es una constante, el problema es *NP-duro*, la demostración se encuentra en el artículo original. Por ello, el algoritmo genético que vamos a implementar nos dará una solución lo más cercana a la óptima posible, pero no obtendremos la solución óptima en la mayoría de los casos.

Buscamos $H = (V_H, E_H)$ tal que:

- H es conexo
- $|V_H \cap Q| \geq |Q| - k$
- $min\_deg(H) \geq \delta_{min}$
- $diam(H)$ se minimiza

Problema 2.1 (ORIGINAL-P2.1): En este caso, la idea es maximizar el mínimo grado de H, utilizando el diámetro como restricción ($diam_{max}$). Este también es un problema *NP-duro*, por lo que la solución obtenida tampoco será la óptima. Pero,

el objetivo es tener una solución cercana a la óptima en un tiempo aceptable.

Buscamos $H = (V_H, E_H)$ tal que:

- H es conexo
- $|V_H \cap Q| \geq |Q| - k$
- $diam(H) \leq diam_{max}$
- $min\_deg(H)$ se maximiza

Problema 2.2 (ORIGINAL-P2.2): El último problema es una propuesta para conseguir una aproximación al resultado óptimo en tiempo polinomial. Es una modificación del problema 2, en este caso en vez de utilizar el diámetro como restricción utilizan un parámetro de distancia máxima. Esa distancia será la máxima permitida entre cada vértice del subgrafo solución y algún vértice de Q.

Buscamos $H = (V_H, E_H)$ tal que:

- H es conexo
- $|V_H \cap Q| \geq |Q| - k$
- $\forall v \in V_H, d(V_H \cap Q, v) := min_{q \in V_H \cap Q} d(q, v) \leq d_{max}$
- $min\_deg(H)$ se maximiza

Dado que el uso del diámetro como restricción puede resultar extremadamente costoso, se ha optado por únicamente desarrollar el algoritmo para el problema 2.2. De esta manera, se elimina la medida del diámetro, una medida con la que solo se podía trabajar con aproximaciones.

1.4. Estructura del documento

En primer lugar, analizando todas las características propias del algoritmo, veremos cómo se ha desarrollado el algoritmo genético para este problema en particular. Posteriormente, veremos qué cambios necesita el algoritmo base dependiendo del problema que se esté afrontando en cada momento. Por último, en la experimentación, veremos cuáles han sido los resultados de todos los algoritmos desarrollados en comparación con los originales.

2. Algoritmos genéticos

Los algoritmos genéticos, así como la mayoría de las técnicas heurísticas y metaheurísticas, se inspiran en procesos físicos de la naturaleza, en este caso, en la propia evolución humana. El contexto en el que trabajan los algoritmos genéticos es el siguiente [4].

Nos encontramos en un **entorno** donde hay una **población** formada por **individuos** con distintos valores de **fitness** (adaptación al entorno), lo que le da una cierta probabilidad de **sobrevivir** y **reproducirse**. Estos rasgos son transmitidos a su descendencia, pudiendo **mutar** ciertas características.

Vamos a ver cuáles son las distintas etapas de un algoritmo genético, y como se aplican en nuestro caso específico.

2.1. Población inicial

Para definir la población inicial, primero debemos conocer el entorno en el que trabajamos, para de esta manera poder establecer un sistema correcto para representar a cada individuo.

El entorno en el que nos encontramos en ambos problemas es el mismo, un grafo $G = (V, E)$ en el que tenemos unos vértices objetivo $Q \subseteq V$. Dado que nuestro objetivo es obtener una comunidad de vértices $V_H \subseteq V$, es decir, H un subgrafo de G, podemos definir a cada individuo como los vértices de G que forman H.

Entonces, cada individuo se representa como un array de longitud $|V|$ en la que cada gen (celda) representa un nodo del grafo G. De esta manera, si la celda contiene un 1 significa que ese nodo de G también está en H, y un 0 en el caso contrario.

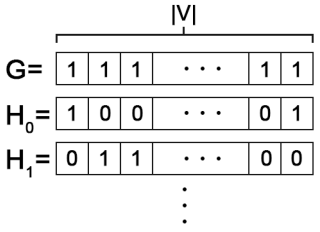


Figura 1: Descripción gráfica de la representación de grafos como individuos.

Para la generación de individuos debemos tener en cuenta el grafo final que buscamos, con el objetivo de intentar que ya los primeros individuos cumplan ciertas condiciones y así asegurar que los resultados se ajustan a lo que pide cada problema. Ambos problemas tienen dos condiciones comunes, que el grafo sea conexo y que contenga al menos $|Q| - k$ vértices de Q.

El método que hemos utilizado para asegurar que se cumplan estas condiciones

es realizar búsquedas en el grafo original G, buscando un número aleatorio de nodos de Q en el rango $\{|Q|-k, ..., |Q|\}$. Las búsquedas que hemos utilizado son: búsqueda en anchura (BFS), búsqueda en profundidad (DFS), A*. Todos ellos modificados para que pare la búsqueda en el momento que encuentra todos los nodos de Q buscados.

La utilización de este método añade complejidad al proceso de creación de la población inicial, pero hemos creído conveniente utilizarlo para comenzar con una población de mayor calidad. Para los casos de BFS y DFS la complejidad es de $\mathcal{O}(n+m)$, en cambio, para el caso del algoritmo A* la complejidad es de $\mathcal{O}(m + nlog(n))$.

2.2. Descendencia

Una vez creada la primera población, esta ha de reproducirse para generar nuevos individuos. Al igual que en la naturaleza, aquí dos individuos se juntan para crear otros nuevos. A esto lo llamamos recombinación. Además, como pasa también en la naturaleza, se pueden producir mutaciones que alteren los nuevos individuos.

2.2.1. Recombinación

En este proceso dos individuos se utilizan para formar otros dos nuevos, juntando siempre a los mejores entre sí. Ambos individuos son cadenas de unos y ceros, que se cortan por un punto aleatorio cualquiera y cada parte se junta con la parte contraria del individuo opuesto. De esta forma se generan dos individuos nuevos.

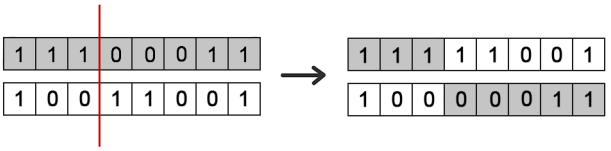


Figura 2: Descripción gráfica del proceso de recombinación de individuos.

En este proceso puede pasar que alguno de los dos individuos generados no cumpla el segundo criterio del problema, es decir, que no tenga tantos nodos de Q como debería. Al saber que previamente ambos tenían los nodos necesarios, sabemos que al menos uno de ellos tendrá más de $|Q| - k$ nodos de Q.

Demostración: Dados dos individuos, los cuales cumplen la restricción, es decir, ambos tienen más de $|Q| - k$ nodos de Q, significa que la suma total de nodos de Q es mayor que $2 \cdot (|Q| - k)$. Por tanto, al sacar dos individuos nuevos mezclando los anteriores, y por ello, dividir los nodos entre ambos, en caso de que un individuo tenga un número de nodos pertenecientes a Q menor que $|Q| - k$ significa que el otro deberá tener más de $|Q| - k$ nodos de Q.

Por ello, si tenemos una población de tamaño p el número de individuos nuevos np será $p/2 \leq np \leq p$.

2.2.2. Mutación

Además, a estos nuevos individuos se les puede aplicar una mutación. El proceso de mutación se refiere a que uno de los genes (celdas) cambia su contenido por azar. Es decir, uno de los números de la cadena cambia al otro número, si es un 0 cambia a 1 y viceversa.

Figura 3: Descripción gráfica del proceso de mutación de individuos.

La mutación se produce con una probabilidad muy baja para no alterar en exceso las generaciones de individuos. Para cada problema veremos que aplicamos la mutación con respecto a ciertas características que ayudarán a la resolución del problema.

2.3. Fitness

La función fitness es la encargada de determinar qué individuos son mejores que otros, es decir, gracias a ella podemos hacer un ranking de los individuos que forman la población. Este ranking nos servirá a la hora de seleccionar a los individuos que pasarán a la siguiente generación y los que serán eliminados.

Para los problemas que vamos a abordar, lo que representa cómo de bueno es el subgrafo resultante es el valor de la función objetivo de cada uno. Por ello, para cada problema el fitness vendrá dado por el valor de la función objetivo. En el problema 1 será el diámetro y en el problema 2 el mínimo grado.

2.4. Selección

Una vez que la población ha generado su descendencia contamos con un mayor número de individuos, de los cuales seleccionaremos tantos como la longitud de la población inicial.

Primero, eliminaremos a los individuos que no cumplan la restricción dada por el problema, y después, a los que sí la cumplen, empezando siempre a eliminar de menor a mayor fitness.

2.5. Implementación

Algorithm 3 Algoritmo genético base

Input: num_iteraciones
Generación de la población inicial;
for $i \leftarrow 1$ **to** *num_iteraciones* **do**
  Recombinación;
  Mutación;
  Fitness;
  Selección;
end

En general, el bucle de este tipo de algoritmos suele realizarse hasta que se cumpla cierto criterio de parada. En este caso, como se busca estudiar temporalmente el algoritmo, se ejecutan un número predefinido de iteraciones. En la experimentación se valorarán distintos valores dependiendo de la complejidad temporal que se busque en cada caso.

Cada problema cuenta con tres restricciones que debe cumplir la población generada por el algoritmo, que son las siguientes:

- H es conexo
- $|V_H \cap Q| \geq |Q| - k$
- Restricción marcada por cada problema: $\delta_{min}, diam_{max}, d_{max}$

Se puede asegurar que la segunda restricción siempre se cumple, ya que, como se ha explicado anteriormente, todo individuo generado, recombinado o mutado siempre contiene al menos $|Q| - k$ vértices de Q.

Sin embargo, la primera de las restricciones no es comprobada en ningún momento, por lo que podríamos estar trabajando con individuos formados por distintas componentes conexas que podrían empeorar ciertos aspectos de la ejecución. Para solucionarlo, cada cierto número de iteraciones se comprueba en cada individuo el número de componentes conexas que tiene, quedándonos solo con las componentes que cumplen la segunda restricción y eliminando el resto.

Esto no se hace en cada iteración, dado que aumentaría bastante el tiempo de ejecución del algoritmo, aparte de que un individuo inconexo también puede generar una buena descendencia. Lo único que no se busca es contar con él como individuo válido a la hora de terminar la ejecución.

Por último, con la implementación explicada hasta ahora no se puede asegurar que la tercera restricción se vaya a cumplir siempre. Por ello, dependiendo de cada problema, se ha de modificar la implementación de una manera u otra. Esto se verá en los próximos apartados.

3. Problema 1

En el primer problema se trata de minimizar el diámetro del grafo resultante utilizando el mínimo grado como restricción. Al estar trabajando con heurísticas estos parámetros serán utilizados a lo largo del proceso.

3.1. Fitness

En los algoritmos genéticos la función que se intenta optimizar es la función fitness, por lo que esta será la encargada de calificar a los individuos en función de su diámetro. Esto hará que en cada iteración se elimine a los individuos con mayor diámetro y de esta manera se vaya alcanzando una mejor solución en cada iteración.

Puede haber casos en los que el diámetro sea igual en más de un individuo, así que, para tener una mejor medida de cuán bueno es cada individuo, la función fitness para este problema también cuenta con el tamaño del grafo. Dado que el tamaño debe ser un factor que aplique a lo que realmente hay que optimizar que es el diámetro la función correcta es realizar el producto entre el diámetro y el tamaño. Pero, al ser el diámetro de mayor importancia que el tamaño, la función fitness que se utilizará para este problema es:

$$fitness(H) = diam(H)^2 \cdot |H|$$

El cálculo del diámetro se aplica mediante el primer algoritmo explicado en la introducción. Por tanto, el coste del cálculo total del fitness es $\mathcal{O}(n + m)$.

3.2. Mutación

Por otra parte, la restricción es la que se aplica a la hora de seleccionar. Cuando tengamos que eliminar a los peores individuos los primeros que se eliminan son los que no cumplen la restricción. El problema es que, al hacer esto, puede darse el caso de que solo se eliminen individuos por la restricción y no contemos con el parámetro importante que es el fitness. Por ello, con la mutación se intenta ayudar a que el mínimo grado sea lo más grande posible.

Cada vértice del grafo tiene una probabilidad de mutación diferente, dependiendo de su grado. Si un vértice tiene grado pequeño, su probabilidad de mutar de 0 a 1, es decir, de no estar en el subgrafo a sí estarlo, es pequeña. En cambio, la probabilidad de mutar de 1 a 0 es grande. En caso de que el grado del vértice sea mayor, la probabilidad de añadirse al subgrafo sube y la de eliminarse del subgrafo disminuye.

Con esto se consigue que la mutación, aparte de buscar individuos nuevos diferentes a los que se pueden crear recombinando, pueda de cierto modo ayudar a que el subgrafo resultante sea más denso.

3.3. Implementación

En este apartado se explica paso a paso el funcionamiento del algoritmo para este problema en concreto, dado que, como se ha mencionado previamente, cada problema necesita de ligeras modificaciones en función de las restricciones.

- *PASO 0*: Modificar el grafo original

Dentro del grafo inicial están contenidos vértices que no tienen el mínimo grado que requiere la restricción, por lo que estos nodos nunca van a ser seleccionables para una posible solución. Para arreglarlo, como paso previo del algoritmo se aplica el "Greedy peeling algorithm" explicado en [5]. Este algoritmo consiste en ir eliminando del grafo los vértices de menor grado hasta que el mínimo grado sea el buscado. Este algoritmo no tiene un número de ejecuciones determinado ya que después de cada eliminación el mínimo grado puede disminuir. Aun así, la complejidad del algoritmo es muy razonable, $\mathcal{O}(m+n)$.

Algorithm 4 "Greedy Peeling Algorithm"

Input: $G = (V, E)$
Output: $S \subseteq V$
$S_n \leftarrow V, i \leftarrow n$;
while i ¿*1* **do**
  $v_{\min} \in \arg\min\{\deg_{S_i}(v) \mid v \in S_i\}$;
  $S_{i-1} \leftarrow S_i \setminus \{v_{\min}\}$;
  $i \leftarrow i - 1$;
end
return $S_{\max} \in \arg\max\{d(S) \mid S \in \{S_1, \ldots, S_n\}\}$

De todas formas, este paso solo se aplica una vez previa al inicio del algoritmo, por lo que no es ejecutado dentro del bucle genético.

- *PASO 1*: Establecer población inicial

El primer paso al comenzar el algoritmo es realizar las búsquedas para obtener la población inicial tal y como han sido explicadas en el apartado anterior. Al crear estos individuos no está garantizado que su mínimo grado sea suficiente, por ello, se realiza un algoritmo de ampliación del individuo.

Este proceso trata de añadir nodos de manera que se verifique la condición del mínimo grado. La idea es añadir a los vértices que no llegan al mínimo grado el número de vértices necesarios para que cumplan la restricción, priorizando seleccionar vértices con más conexiones con el individuo actual. La complejidad temporal es, a lo sumo $\mathcal{O}(m+n)$.

Algorithm 5 Completar el grafo por la restricción del mínimo grado

Input: $G = (V, E), \delta_{min}$
Output: H
$H \leftarrow G$
while $\delta_{min} > min\_deg(H)$ **do**
  $U \leftarrow \{v \in V \mid deg(v) < \delta_{min}\}$
  $A \leftarrow []$
  for n *in* U **do**
    $deg \leftarrow deg(n)$
    $N \leftarrow \{v \in V \mid (n, v) \in E\}$
    for e *in* N **do**
      if $deg >= \delta_{min}$ **then**
        break
      end
      if $e \notin (V \cup U)$ **then**
        $A \leftarrow e$
        $deg \leftarrow deg + 1$
      end
    end
  end
  $H \leftarrow H \cup (A, E_A)$
end
return H

Una vez que el individuo está formado y cumple todas las restricciones se calcula su fitness, con complejidad de $\mathcal{O}(m + n)$. Y está listo para comenzar el bucle genético.

- *PASO 2*: Nuevos individuos

El siguiente paso es crear nuevos individuos a partir de los que ya existen. Estos se crean a través de los procesos de recombinación y mutación. Además, a los individuos creados se les ha de calcular el fitness.

Dentro de este paso, en una de cada 5 iteraciones, además de en la última iteración, se realiza también la comprobación de la conectividad y el aumento del número nodos para asegurar que los individuos cumplen todas las restricciones.

- *PASO 3*: Selección

Una vez que los nuevos individuos están creados se comienza el proceso de selección. En este paso se eliminan los peores individuos y los mejores son los que pasan a la siguiente iteración del bucle.

- *PASO 4*: Volver al *PASO 2*, o, en caso de terminar iteraciones, finalizar la ejecución

3.4. Complejidad

El número de iteraciones del bucle para este problema es de $n^{0,25}$, por lo que la complejidad total es la siguiente:

$$\mathcal{O}(m + n) + \mathcal{O}(2 \cdot (n + m)) + \mathcal{O}(n^{0,25} \cdot (1{,}2 \cdot (n + m))) = \mathcal{O}(n^{0,25} \cdot (n + m))$$

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

4. Problema 2

Este otro problema trata de maximizar el mínimo grado utilizando como restricción la distancia de los nodos añadidos a los nodos de Q que estén presentes en el grafo. Este es una modificación del problema 2 en el que se prescinde del diámetro, una medida que puede ser problemática. Pese a que el algoritmo propuesto para este problema devuelve la solución exacta en tiempo polinomial, se propone un algoritmo genético alternativo.

4.1. Fitness

En este problema el fitness viene marcado por el mínimo grado, una medida que, al igual que en el problema 1, puede darse el caso de dos individuos con mismo mínimo grado pero muy distintos entre sí. Por ello, esta medida también debe ser modificada para un mejor funcionamiento del algoritmo. Como medida auxiliar se utiliza, de nuevo, el número de nodos, aunque esta vez de manera distinta.

La idea es que el número de nodos sea un factor que permita diferenciar a dos individuos con el mismo mínimo grado, pero nunca dar prioridad a un individuo que tenga menor grado que otro solamente por su tamaño. Para ello, al mínimo grado se le añade el tamaño del individuo normalizado con valores entre 0 y 1, siendo 1 el grafo original. Aunque, al tener que maximizar, lo que buscamos es el complementario de dicha normalización.

$$fitness(H) = min\_deg(H) + (1 - \frac{|H|}{|G|})$$

Al contrario que en el problema 1, el cálculo de esta medida no tiene un coste temporal significativo, por lo que no se necesitará de ningún método de aproximación para calcularla.

4.2. Mutación

La mutación se realizará con el mismo sistema que en el problema 1, por lo tanto, en vez de ser una ayuda con la restricción será una ayuda para la medida a optimizar, el mínimo grado.

4.3. Implementación

Al igual que en el resto de problemas, la restricción impone una complicación a la hora de desarrollar el algoritmo. En este caso se debe comprobar que cada individuo tenga todos los nodos a una distancia menor que d_{max} de los nodos de Q que contenga.

Existen varias formas de afrontar este problema. El objetivo es reducir al máximo la complejidad temporal, por ello, se ha de reducir el número de veces que se comprueba si un nodo cumple las características marcadas. Para ello se ha optado por el siguiente procedimiento.

- *PASO 0*: Obtener los nodos posibles dado Q

Al inicio de la ejecución se comprueba para cada nodo $q \in Q$ qué nodos están a una distancia menor o igual a la restricción, es decir, qué nodos son válidos para que puedan pertenecer a los individuos finales. Estas listas de nodos se guardan en el mismo formato que los individuos, en cadenas de ceros y unos. En este caso, los unos representan los nodos v tal que $d(q, v) \leq d_{max}$.

Algorithm 6 Reducir grafo por la restricción de distancia máxima

Input: $G = (V, E), Q \subseteq V, d_{max}$
Output: H
$H = \{(u, v) \in E \mid u \in V, v \in V, \text{dist}(u, Q) \leq d_{\max}, \text{dist}(v, Q) \leq d_{\max}\}$
return H

Al mismo tiempo, se han identificado todos los nodos que son seleccionables, dado que ningún elemento que no esté a una distancia menor a d_{max} de algún nodo de Q nunca va a formar parte de una solución. Por ello, el grafo se reduce, quedando únicamente los nodos alcanzados, reduciendo el número de nodos, el número de aristas, y, por consiguiente, el tiempo de ejecución.

Este paso tiene una complejidad de $\mathcal{O}(|Q| \cdot (m + n)$, y gracias a los datos obtenidos no se necesitará realizar comprobaciones tan costosas dentro del bucle genético.

- *PASO 1*: Establecer población inicial

Mismo procedimiento que en el problema 1, se generan los individuos que forman la primera población. En este caso en cálculo del fitness se ve reducido dado que la complejidad para obtener el mínimo grado es menor.

- *PASO 2*: Nuevos individuos

Se aplican los procesos de recombinación y mutación ya explicados, y en una de cada 10 iteraciones se realiza la comprobación de la restricción del problema, además de comprobar la conectividad de los individuos.

Para comprobar que se cumple la restricción, y, en caso contrario, modificar los individuos para que la cumplan, se emplearán los "individuos" obtenidos en el *PASO 0*.

Primero se suman todos los individuos asociados a cada $q \in Q$ siempre que $q \in V_H$, siendo H cada individuo de la población. De esta manera, se obtiene una lista con todos los nodos que cumplen la restricción de la distancia. Posteriormente, se multiplica la cadena de cada individuo por esta cadena formada. Esto devuelve la cadena

del nuevo individuo, el cual cumple la restricción al haber sido eliminados los nodos que no la cumplían.

La complejidad de este paso es $\mathcal{O}(|Q| \cdot n) + \mathcal{O}(n)$. Dado que $|Q|$ es, o suele ser, muy pequeño se podría decir que la complejidad total es $\mathcal{O}(n)$.

- *PASO 3*: Selección

Eliminar a los peores individuos y mantener a los mejores.

- *PASO 4*: Volver al *PASO 2*, o, en caso de terminar iteraciones, finalizar la ejecución

4.4. Complejidad

El número de iteraciones para este problema es el mismo que para el problema 1, es decir, $\sqrt[4]{n}$.

$\mathcal{O}(|Q| \cdot (m+n) + \mathcal{O}(m+n) + \mathcal{O}(n^{0,25} \cdot ((m+n) + 0{,}1 \cdot (|Q| \cdot n)) = \mathcal{O}(n^{0,25} \cdot ((m+n) + 0{,}1 \cdot (|Q| \cdot n)) = \mathcal{O}(n^{1,25} \cdot |Q|)$

5. Experimentación

Como se ha mencioado previamente, el objetivo de este trabajo es resolver los problemas propuestos en [1] con una alternativa a la que se expone en dicho artículo. Mediante el uso de algoritmos genéticos, la idea es comparar los resultados de ambos algoritmos y determinar cuál de los dos puede ser más beneficioso dependiendo de las conciciones en las que se trabajen.

En este apartado se van a replicar algunas de las pruebas realizadas en [1], que tienen como principales objetivos analizar los siguientes aspectos:

- La capacidad del algoritmo para detectar outliers;
- Clasificación de outliers en comparación con el subgrafo de mínima ineficacia (MIS) definido en [7];
- Diferencias entre las soluciones que ofrece el algoritmo para distintos valores de k y de las restricciones;
- Diferencias entre las soluciones que ofrece el algoritmo dependiendo de el conjuntos de vértices Q que se utilize y de la distancia entre sus vértices;
- Tiempos de ejecución.

De estas pruebas se va a prescindir de una, "Diferencias entre las soluciones que ofrece el algoritmo dependiendo de el conjuntos de vértices Q que se utilize y de la distancia entre sus vértices", dado que las conclusiones que se sacan en el artículo original hacen ver que no se obtienen resultados realmente informativos, ya que llegan a la conclusión de que no afecta al resultado del algoritmo.

Además, se añaden dos apartados adicionales a la experimentación:

- Análisis de los grafos utilizados
- Comparación del resultado con respecto al resultado óptimo

5.1. Análisis de los grafos utilizados

Primero de todo, es conveniente analizar cada uno de los grafos que van a ser utilizados para la experimentación. Esto nos servirá para identificar cuáles son los casos en los que un algoritmo trabaja mejor que el otro, o puede proporcionar pistas sobre el por qué de los resultados obtenidos.

Se va a trabajar con tres grafos distintos, todos ellos provienen del Stanford Large Network Dataset Collection (SNAP), de la universidad de Stanford. De todos ellos se nos proporciona tanto el grafo en sí, como una lista de las 5000 comunidades más grandes.

	Amazon	Youtube	DBLP
Vértices	334863	1134890	317080
Aristas	925872	2987624	1049866
Diámetro	44	20	21
Densidad	$1{,}6 \times 10^{-5}$	$4{,}6 \times 10^{-6}$	$2{,}1 \times 10^{-5}$

TABLA 1: Características de los grafos en la experimentación. Se muestra el nombre del dataset, el número de vértices, el número de aristas, el diámetro del grafo y la densidad.

En la tabla 1 se muestran algunas de las principales características de los grafos provenientes de Amazon, Youtube y DBLP.

El dataset de Amazon representa relaciones entre productos. Si un producto suele ser comprado a la vez que otro, existe una arista que conecta ambos nodos. En el caso de DBLP los nodos representan investigadores, en caso de que dos investigadores hayan trabajado juntos en un artículo, revista o conferencia, estarán conectados. Por último, el grafo de Youtube representa las amistades entre los usuarios de la red social.

Se puede apreciar como el grafo de Amazon tiene un grado de dispersión mucho mayor al resto, ya que cuenta con un bajo número de vértices y un diámetro mayor que el doble de los demás. En cambio, el grafo más denso es el de Youtube, ya que cuenta con más del triple de nodos que los otros dos casos y sin embargo tiene la mitad de diámetro que el de Amazon y similar al de DBLP. Esto puede hacer que, a la hora de buscar comunidades, las comunidades del grafo de Youtube estén más solapadas, y esto pueda interferir en la ejecución del algoritmo.

Para cada uno de estos grafos contamos con una lista de las mayores comunidades. También es necesario analizar las características de estas comunidades, para poder examinar los patrones del comportamiento de ambos algoritmos en función de las comunidades.

	Amazon	Youtube	DBLP
Media	13,49	14,49	22,44
Moda	3	2	6
Mediana	8	4	8
Varianza	17,51	60,45	201,06

TABLA 2: Características de las 5000 mayores comunidades de cada grafo. Se muestran la media, moda, mediana y varianza del tamaño de cada comunidad.

Como se puede apreciar, las comunidades de cada grafo tienen tamaños completamente distintos. Para empezar, Amazon cuenta con las comunidades más pequeñas, ya que tanto la media de los tamaños como la varianza son menores que en el resto de casos.Youtube, sin embargo, tiene una varianza en comparación mucho mayor, con una mediana muy baja, lo que hace ver que la mayor parte de las comunidades son muy pequeñas pero después hay otras muy grandes que compensan la media. Por último, el caso de DBLP es el más radical, llegando a una varianza 100 veces

mayor que en el caso de Amazon pese a tener ambos grafos más o menos la misma estructura.

5.2. Detección de outliers

Grafo	Algoritmo	$k = 1$	$k = 2$	$k = 3$	$k = 4$
DBLP	ORIGINAL-P1 $[\delta_{min} = 3]$	0,3 (30 %)	0,8 (40 %)	1,2 (40 %)	1,8 (45 %)
Amazon		–	0,7 (35 %)	0,9 (30 %)	1,4 (35 %)
Youtube		0,5 (90 %)	1 (50 %)	1,5 (50 %)	2 (50 %)
DBLP	GENÉTICO-P1 $[\delta_{min} = 3]$	1 (100 %)	2 (100 %)	3 (100 %)	4 (100 %)
Amazon		1 (100 %)	2 (100 %)	3 (100 %)	4 (100 %)
Youtube		0,9 (90 %)	2 (100 %)	2,7 (90 %)	3,7 (92,5 %)
DBLP	ORIGINAL-P2.1 $[diam_{max} = 4]$	1 (100 %)	1,9 (95 %)	2,8 (93,3 %)	3,8 (95 %)
Amazon		0,9 (90 %)	1,9 (95 %)	2,9 (96,7 %)	3,7 (92,5 %)
Youtube		0,6 (60 %)	1,3 (65 %)	1,6 (53,3 %)	3,73 (93 %)
DBLP	ORIGINAL-P2.2 $[d_{max} = 2]$	0,5 (50 %)	1,3 (65 %)	1,8 (60 %)	2,7 (67,5 %)
Amazon		0,9 (90 %)	1,9 (95 %)	2,9 (96,7 %)	3,9 (97,5 %)
Youtube		0,4 (40 %)	0,6 (30 %)	0,9 (30 %)	1,4 (35 %)
DBLP	GENÉTICO-P2 $[d_{max} = 2]$	1 (100 %)	1,9 (95 %)	3 (100 %)	4 (100 %)
Amazon		1 (100 %)	2 (100 %)	3 (100 %)	4 (100 %)
Youtube		0,9 (90 %)	2 (100 %)	2,6 (86,7 %)	4 (100 %)

TABLA 3: Capacidad de identificación de valores atípicos de los dos algoritmos aplicados para el problema 1 en tres conjuntos de datos, con $n = 10$ y $m = k \in [1, 4]$. La tabla muestra el número medio de valores atípicos detectados (y el porcentaje con respecto al número total de valores atípicos presentes).

En la primera fase de la experimentación se busca estudiar la capacidad del algoritmo de detectar y descartar outliers dentro del conjunto Q. Para esta prueba se seleccionan n vértices de una misma comunidad de las 5000 que se conocen. Después, se seleccionan m elementos aleatorios que no formen parte de esa comunidad. El conjunto formado por todos ellos es el que se pasa al algoritmo, con un valor para $k = m$. Esta prueba se repite para los valores de $m = [1, 4]$.

La idea es que después de la ejecución los valores atípicos desaparezcan de la solución que proporcione el algoritmo. Para los resultados que se muestran en 3 se han generado un total de 10 pruebas diferentes, obteniendo un valor medio de outliers detectados que es el que se encuentra en la tabla. Para los resultados del algoritmo original se realizaron 20 pruebas distintas, obteniendo el resultado haciendo la media de las 10 primeras ejecuciones exitosas (ya que el otro algoritmo puede no dar solución factible). En caso de que no se alcancen dichas ejecuciones exitosas el resultado cuenta como nulo.

Como se puede apreciar, el algoritmo genético del problema 1 tiene un éxito prácticamente del 100 % a la hora de encontrar valores atípicos, aparte de la ventaja de que siempre se obtenga una solución, ya sea cercana a la óptima o no. En cambio, con el algoritmo original rara vez se alcanza el 50 % de outliers detectados y eliminados.

Por otra parte, el resto de algoritmos tienen un porcentaje de acierto mucho mayor al primero, pero siguen sin llegar a la efectividad del genético 1.

El algoritmo genético propuesto para el problema 2 obtiene resultados algo peores, pero dadas las pocas muestras utilizadas para la comprobación se podrían considerar iguales. Por ello, al igual que el otro algoritmo, consigue resultados mucho mejores que el algoritmo que replica y que el resto de rivales.

5.3. Clasificación de outliers en comparación con MIS

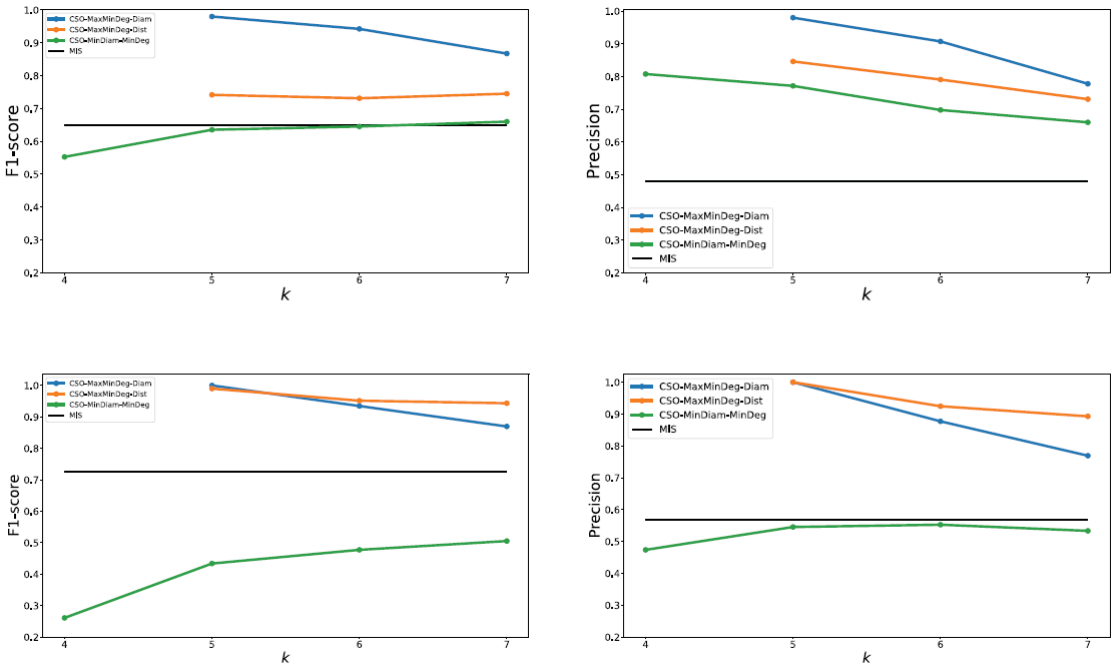
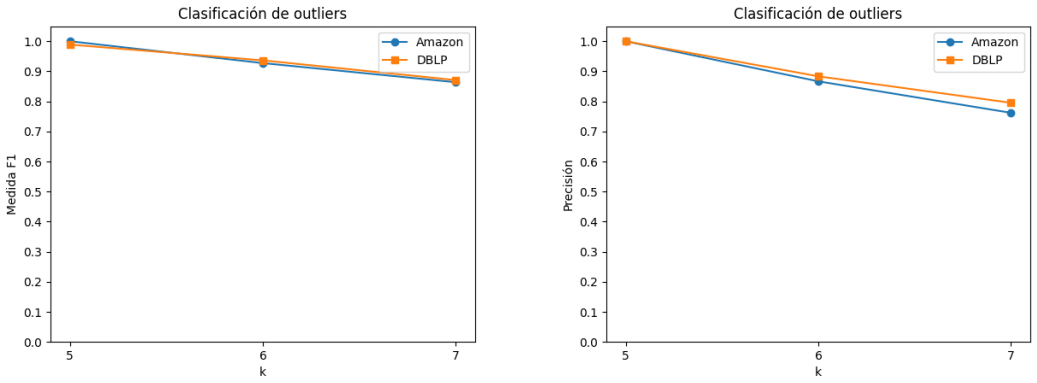
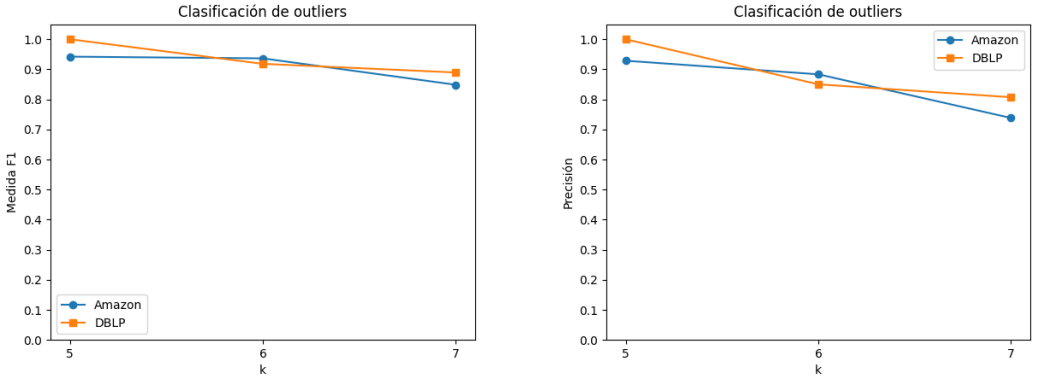


Figura 4: Resultados obtenidos del artículo original, mostrando el F1-score medio y la precisión media para los grafos de *DBLP* (arriba) y *AMAZON* (abajo).



(a) Medida F1 (b) Precisión

Figura 5: Resultados obtenidos por el algoritmo genético respectivo al problema 1 para los grafos de *AMAZON* y *DBLP* con $k \in 5, 6, 7$ y $\delta_{min} = 3$.



(a) Medida F1 (b) Precisión

Figura 6: Resultados obtenidos por el algoritmo genético respectivo al problema 2 para los grafos de *AMAZON* y *DBLP* con $k \in 5, 6, 7$ y $d_{max} = 2$.

Esta fase trata de analizar la detección de outliers de una manera distinta. La idea es considerar cada elemento de Q como "valor atípico" o "valor no atípico" y analizar el algoritmo como si se tratase de un modelo de clasificación de outliers. Además, los algoritmos serán comparados con el algoritmo MIS estudiado en [7].

Este problema básicamente recibe $Q \subseteq V$ y $G = (V, E)$ y trata de obtener un subgrafo con cierta cohesión que contenga los vértices de Q, pero no necesariamente conecta a todos. Por ello, para analizar los resultados se coge la mayor componente conexa de la solución obtenida por este algoritmo.

El procedimiento es el siguiente, se seleccionan $n = 10$ elementos de una misma comunidad, con una distancia entre ellos de a lo sumo 4, y $m = 5$ elementos de fuera de esa comunidad. El algoritmo se ejecuta 10 veces para valores de $k \in [4, 7]$

y $\delta_{min} = 3$, obteniendo como resultado el valor medio de todas las pruebas.

Las medidas utilizadas en este apartado son el F1 y la precisión, por lo que se ha de rellenar la siguiente matriz de confusión.

		Resultado	
		outlier	no outlier
Q	outlier	Verdadero positivo (VP)	Falso negativo (FN)
	no outlier	Falso positivo (FP)	Verdadero negativo (VN)

TABLA 4: Matriz de confusión

$$\text{Precisión} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Positivos}}$$

$$\text{Recall} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Negativos}}$$

$$F1 = \frac{2 \cdot \text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}}$$

Como se puede apreciar en la figura 5, en el problema 1 todos los resultados son ampliamente mejores que los obtenidos por el algoritmo al que replican (ORIGINAL-P1). En cambio, comparando con el resto de algoritmos, obtiene resultados similares al mejor algoritmo en cada caso. Por ejemplo, el algoritmo ORIGINAL-P2.1 parece funcionar mejor para el grafo de *DBLP*, mientras que el algoritmo ORIGINAL-P2.2 obtiene mejores resultados con el grafo de *Amazon*. Sin embargo, el algoritmo genético tiende a igualar el mejor resultado de ambos algoritmos.

Por otro lado, los resultados obtenidos por el algoritmo genético desarrollado para el problema 2 parecen tener una calidad cercana al del problema 1. Aunque se puede apreciar que no hay un grafo en el que se comporte claramente mejor que en el otro, cuando en los resultados respectivos al problema 1 sí se distingue una clara tendencia de tener un peor resultado en el grafo de *Amazon*.

Observando las diferencias entre ambos grafos 1 esto puede dar pistas sobre en qué tipo de grafos podría funcionar mejor este algoritmo. No obstante, tanto el número de ejecuciones como la diferencia de resultados no son suficientes como para asegurar que pueda haber una diferencia real de rendimiento dependiendo de las características del grafo.

Otro aspecto perfectamente reconocible dentro de ambos casos es la tendencia de las medidas a disminuir con respecto al aumento de k. El motivo es evidente, con el fin de disminuir el tamaño, y así conseguir una comunidad más pequeña y compacta, siempre se va a intentar rechazar el máximo número de nodos posibles. Esto hace que crezca el número de Falsos Negativos y, por tanto, disminuyan tanto la precisión como la medida F1.

La conclusión que se puede obtener de esta fase es que ambos algoritmos genéticos tienden a rechazar la mayor cantidad de nodos posibles, incluso llegando a eliminar nodos que deberían estar en la solución final.

5.4. Soluciones dependiendo de los valores de k y de las restricciones

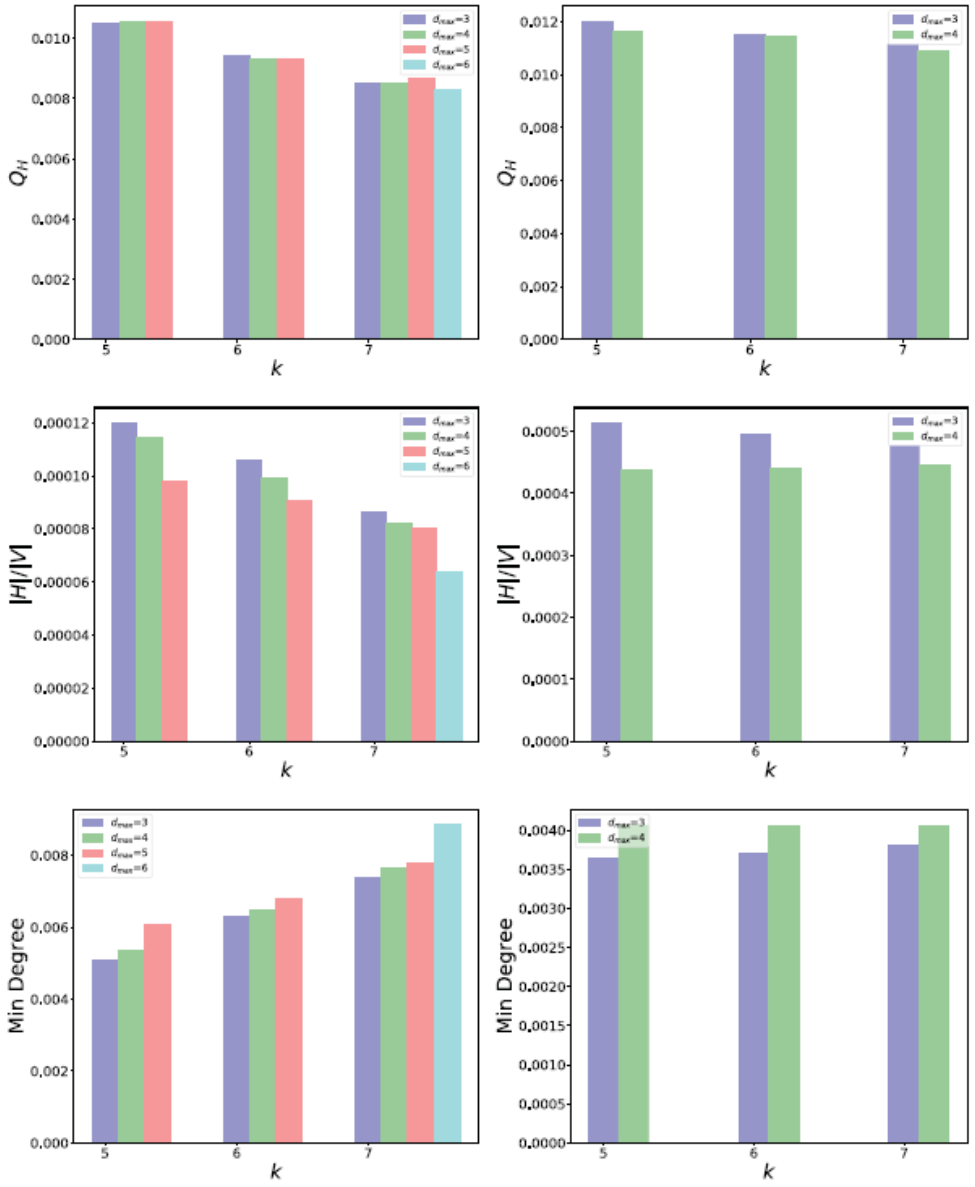


Figura 7: Características de la solución en Youtube (izquierda) y Amazon (derecha) para ORIGINAL-P1 con $k \in \{5, 6, 7\}$ y el grado mínimo δ_{min} en el intervalo $[3, 6]$. Todas las tabla están en formato $1e-3$.

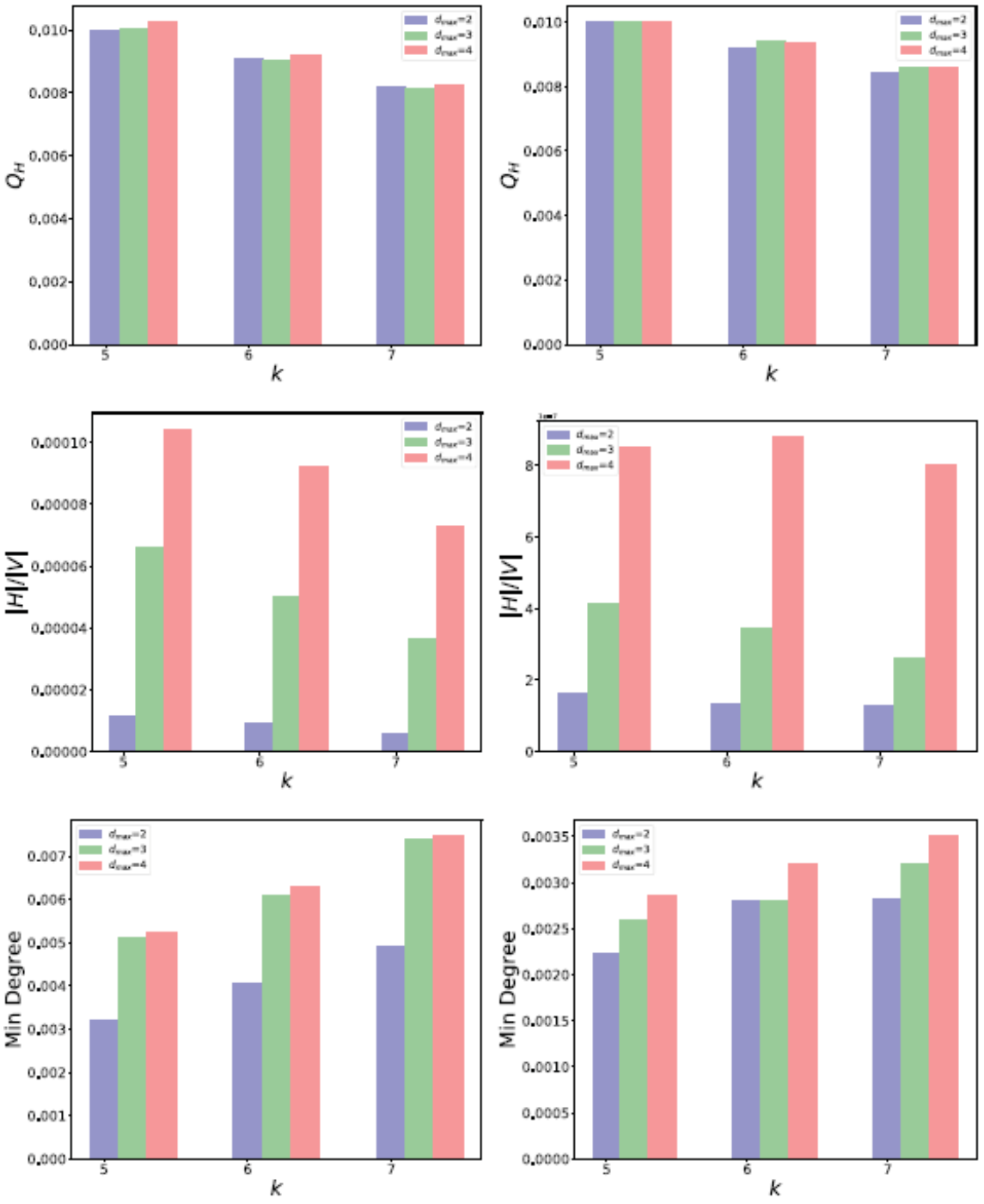


Figura 8: Características de la solución en Youtube (izquierda) y Amazon (derecha) para ORIGINAL-P2.1 con $k \in \{5, 6, 7\}$ y el diámetro mínimo $diam_{max}$ en el intervalo $[2, 4]$. Todas las tabla están en formato $1e-3$.

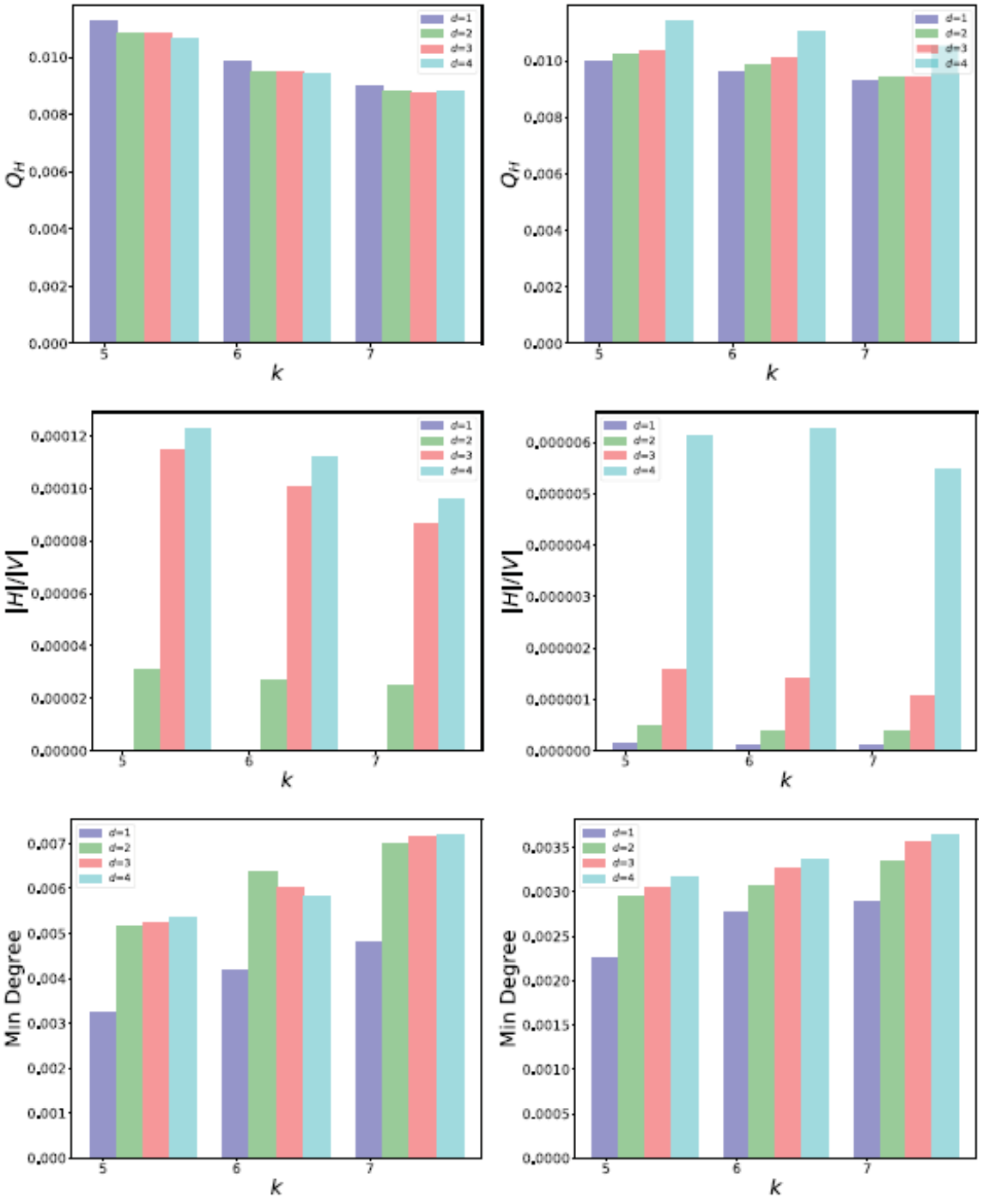


Figura 9: Características de la solución en Youtube (izquierda) y Amazon (derecha) para ORIGINAL-P2.2 con $k \in \{5, 6, 7\}$ y la distancia máxima d_{max} en el intervalo $[1, 4]$. Todas las tabla están en formato $1e-3$.

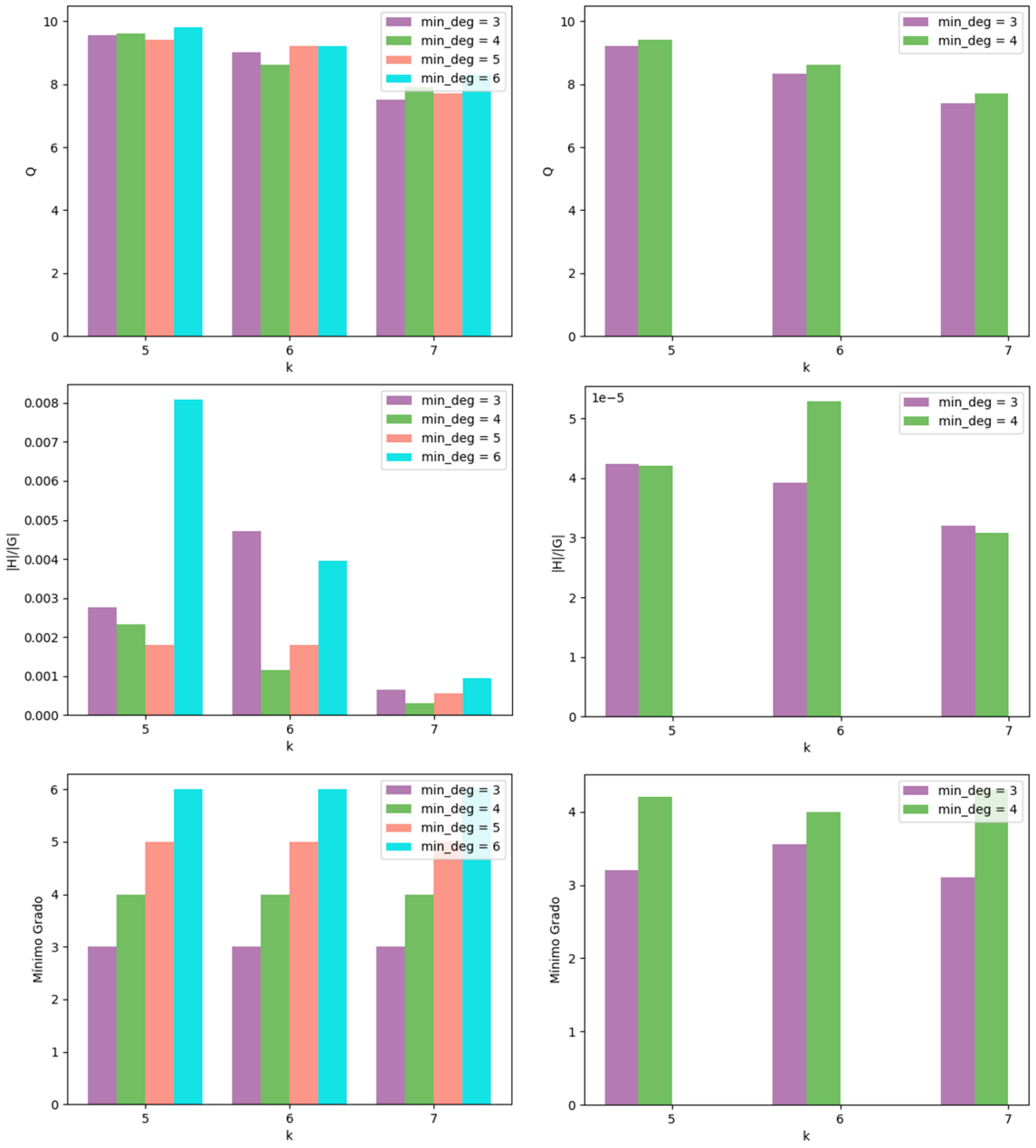


Figura 10: Características de la solución en Youtube (izquierda) y Amazon (derecha) para el algoritmo genético respectivo al problema 1 con $k \in \{5, 6, 7\}$ y el grado mínimo δ_{min} en el intervalo $[3, 6]$.

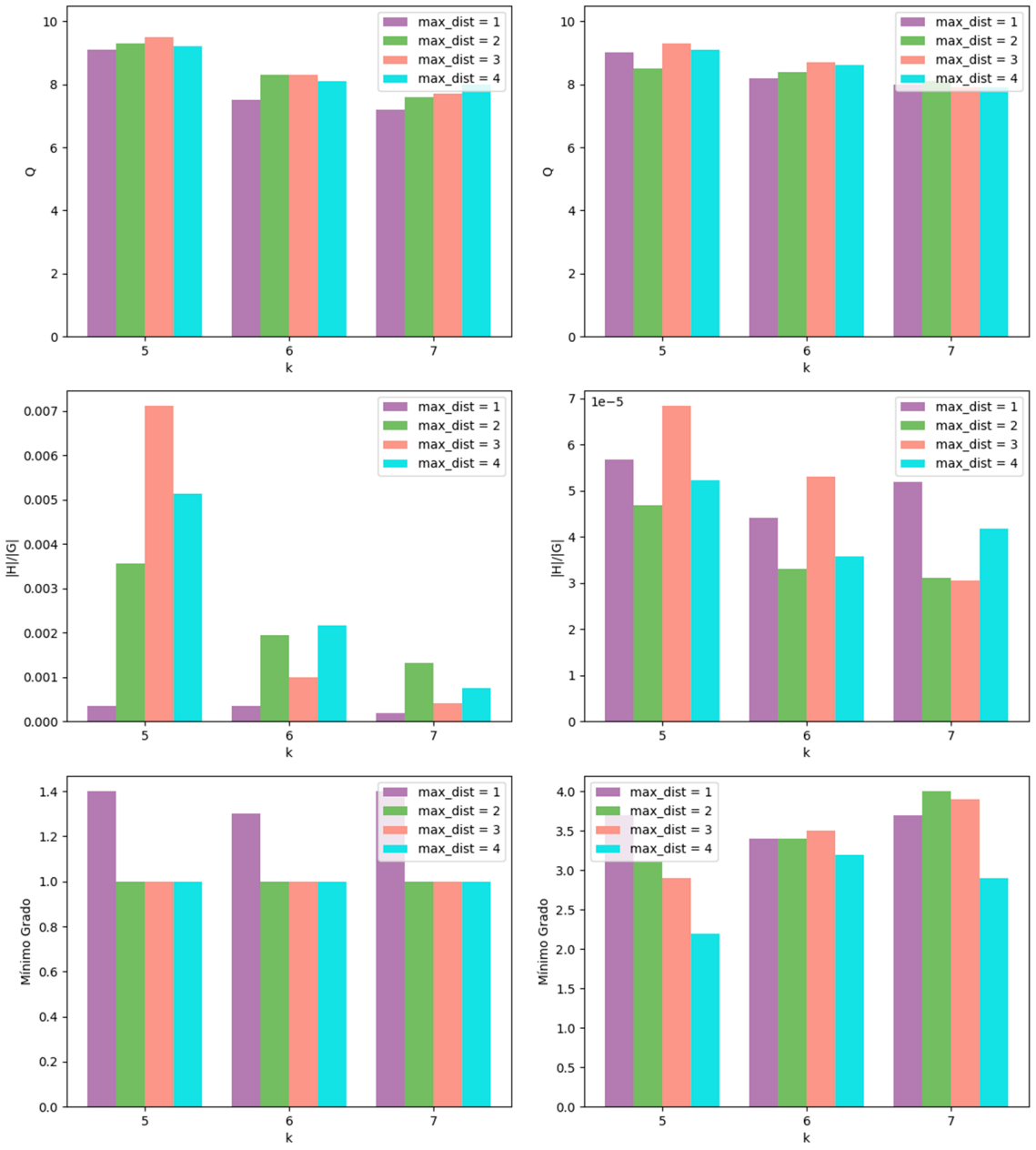


Figura 11: Características de la solución en Youtube (izquierda) y Amazon (derecha) para el algoritmo genético respectivo al problema 2 con $k \in \{5, 6, 7\}$ y la distancia máxima d_{max} en el intervalo $[1, 4]$.

La siguiente fase de la experimentación trata de analizar la precisión de los algoritmos con respecto a los distintos valores de k y de las restricciones. El procedimiento de esta prueba es similar al de la primera fase, se seleccionan aleatoriamente $n = 10$ vértices de la misma comunidad, con la excepción de que debe haber una distancia entre ellos de 4 como máximo. Después, se seleccionan $m = 4$ vértices que no pertenezcan a dicha comunidad. Una vez que se tiene el conjunto de vértices se realizan 10 ejecuciones, de las cuales se analizarán las medidas siguientes:

- El número de vértices de Q contenidos en el subgrafo resultante H
- El número de vértices con respecto al total, $|V_H|/|V_G|$
- El mínimo grado, $min\_deg(H)$

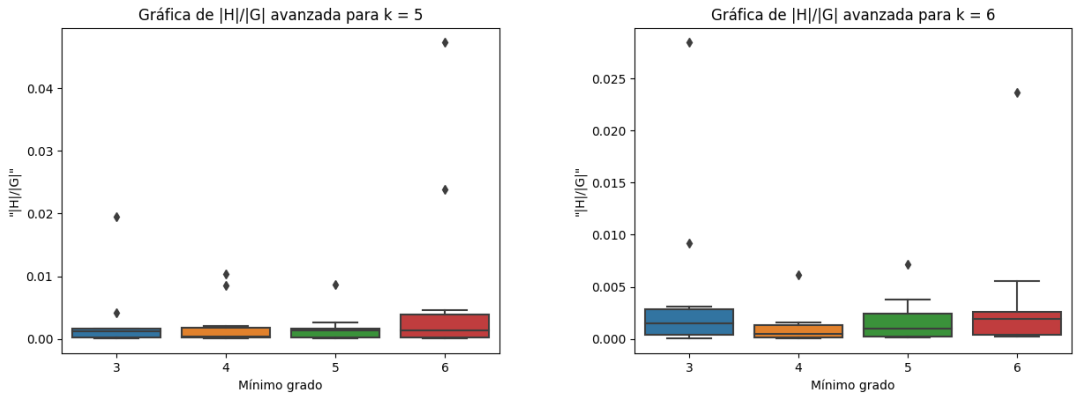
En cuanto a la primera de las medidas, los vértices de Q que se encuentran en la solución de cada algoritmo, se puede apreciar como todas las gráficas tienen una tendencia similar. A menor k, mayor es el número de vértices de Q, esto es algo que ya se ha visto en anteriores fases de la experimentación. Sin embargo, en las gráficas 7 y 9, hay algunas ocasiones en las que el número de vértices de Q supera el óptimo, que es 10. En cambio, el algoritmo genético tiende a quedarse algo por debajo de 10 incluso para el menor k. Esto se debe a que, al ser $m < k$, el algoritmo siempre intenta descartar el máximo número de nodos posibles de manera que la solución tenga el menor diámetro posible.

Analizando el mínimo grado de los resultados obtenidos por los algoritmos genéticos, se aprecian resultados que llaman la atención comparándolos con los de sus algoritmos referencia. En primer lugar, los resultados del primer algoritmo 10 se ajustan exactamente a la restricción en todas las ejecuciones, mientras que el algoritmo 6a tiende a tener valores mayores que la restricción. Es difícil valorar si esto significa una ventaja o un inconveniente a la hora de alcanzar un buen resultado, pero es algo significativo, ya que en ninguna de las soluciones se ha alejado del valor marcado por la restricción.

Por otro lado, en el segundo algoritmo, pese a estar maximizando el mínimo grado, en los resultados de *Amazon* se queda bastante lejos de los valores obtenidos por el algoritmo original. Esta diferencia es aún mayor en el caso de *Youtube*, el cual obtiene mínimo grado de valor 1 la mayoría de casos, pese a utilizar el mínimo grado como medida a maximizar. Es algo que llama la atención, pero no por ello los resultados son peores.

A pesar de que las medidas analizadas hasta ahora son fundamentales para comprobar el correcto funcionamiento del algoritmo, la medida que relaciona el tamaño de la solución con el del grafo completo parece ser la más importante en cuanto a medir la calidad de las soluciones. Para el primer problema, comparando con el algoritmo que intenta replicar, en la figura 7 se aprecia una clara mejoría en cuanto al tamaño de la solución, siendo esta más de 10 veces menor. En cambio, comparándolo con el resto de algoritmos, los resultados son algo más parejos, aunque aparentemente siguen mejorando su desempeño.

Con el punto de mira directamente en la gráfica que analiza el tamaño de la solución del algoritmo genético, se aprecian algunos valores que parecen salirse de la tendencia normal. Como en estas gráficas se muestra únicamente el valor medio de los tamaños de las soluciones, en este caso es conveniente utilizar gráficas adicionales más informativas.



(a) $k = 5$ (b) $k = 6$

Figura 12: GRÁFICAS QUE MUESTRAN LA DISTRIBUCIÓN DEL TAMAÑO DE LAS SOLUCIONES DEL ALGORITMO GENÉTICO.

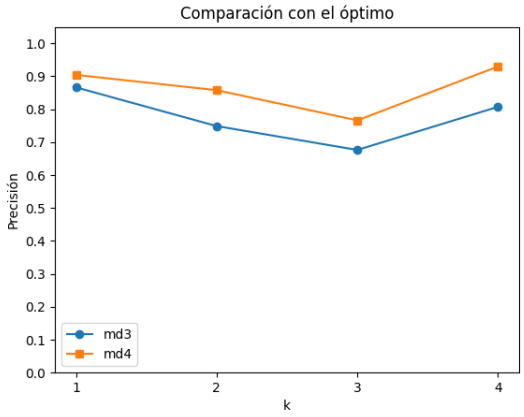
El caso más llamativo es con $k = 5$ y $\delta_{min} = 6$, que rompe por completo la dinámica del resto de valores con la misma k. Analizando los resultados de la figura 12a se ve cómo, aunque los tamaños sí parecen ser algo mayores en general, la fuente del problema se encuentra en dos outliers que tienen más de 8 y 16 veces más tamaño que la mediana. Siendo una muestra tan pequeña (10 ejecuciones) está claro que los resultados anteriores, representados en la figura 8, estaban claramente condicionados por esas dos muestras.

Otro caso significativo es cuando $k = 6$, que no parecen seguir un patrón fijo y en los casos de $\delta_{min} \in \{5, 6\}$ los resultados son bastante peores que en el resto de casos. Se puede apreciar en la figura 12b como ambos casos también están afectados por ejecuciones aisladas que empeoran significativamente los resultados.

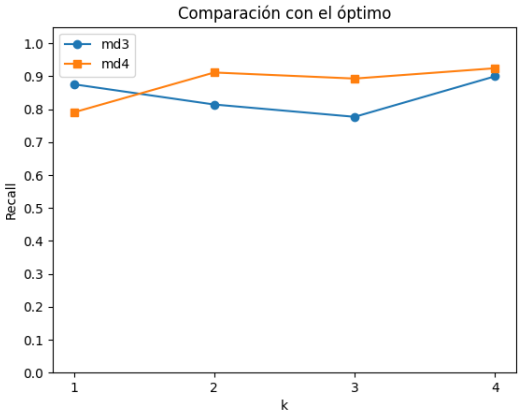
El por qué de estas malas ejecuciones, todas en el grafo de Youtube, se puede explicar a través de la distribución de los tamaños de las comunidades de dicho grafo 2. En el caso de Youtube, sus comunidades tienen una grandísima varianza en comparación con Amazon. Dado que lo que se mide es únicamente el tamaño de la solución, no se puede decir que esos casos aislados sean errores, sino que simplemente están buscando comunidades mucho más grandes que la media, por lo que la solución será en proporción mucho mayor.

Los resultados respectivos al algoritmo del segundo problema son bastante similares a los del primero. Por tanto, siguen mejorando los resultados del artículo, aunque esta vez en menor medida comparado con su algoritmo de referencia. Aunque con la pequeña cantidad de ejecuciones no se pueden sacar muchas conclusiones con diferencias tan pequeñas, parece que este algoritmo funciona mejor para el grafo de *Youtube* y peor para el de *Amazon*. Esto podría relacionarse con las características de cada grafo, pero, como se ha mencionado previamente, no se pueden sacar conclusiones con tan pocas ejecuciones.

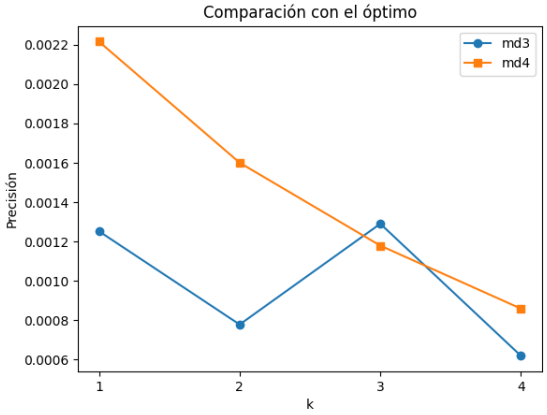
5.5. Comparación con el óptimo



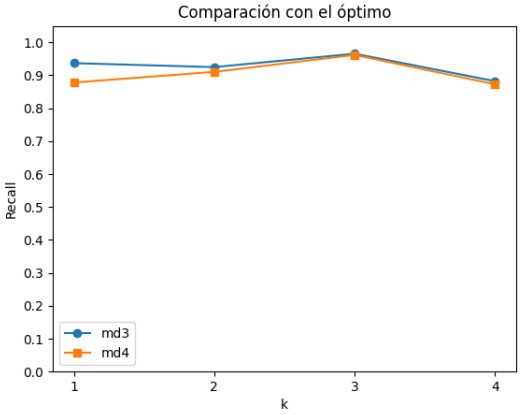
(a) Precisión



(b) Recall

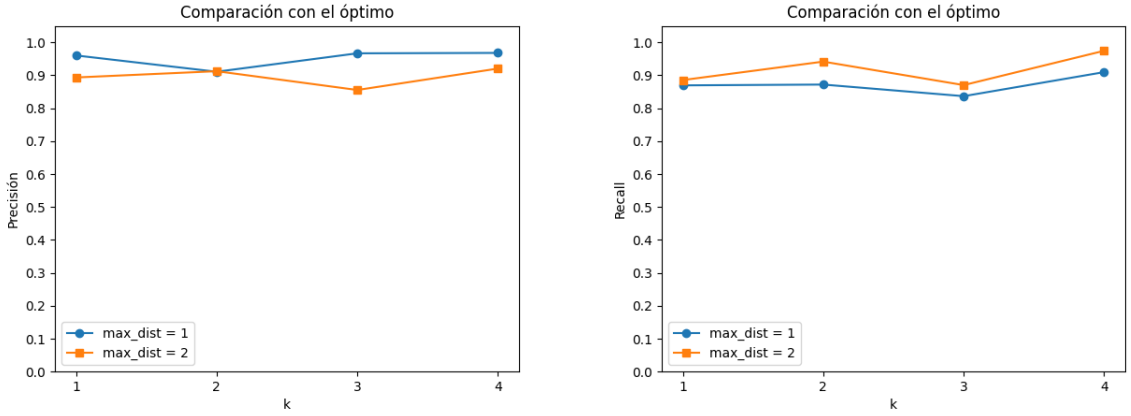


(c) Precisión



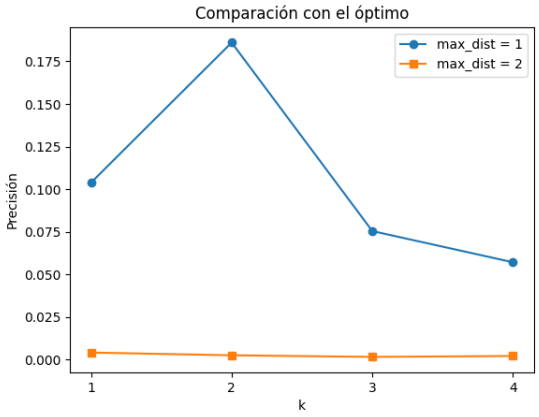
(d) Recall

Figura 13: Gráficas que muestran la precisión y el recall para el grafo de Amazon (arriba) y YouTube (abajo), para los valores de $k \in 1, 2, 3, 4$ y $\delta_{min} = [3, 4]$ del algoritmo genético propuesto para el problema 1.

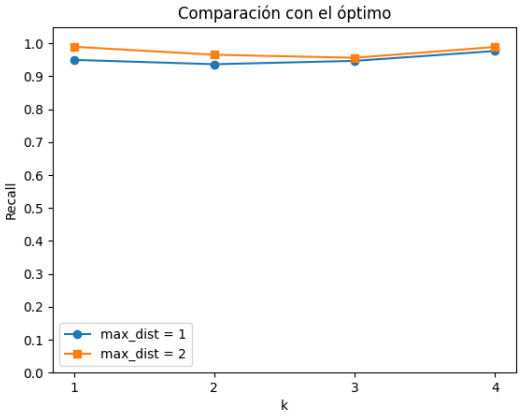


(a) Precisión

(b) Recall



(c) Precisión



(d) Recall

Figura 14: Gráficas que muestran la precisión y el recall para el grafo de Amazon (arriba) y YouTube (abajo), para los valores de $k \in 1, 2, 3, 4$ y $d_{max} = [1, 2]$ del algoritmo genético propuesto para el problema 2.

El conjunto de pruebas realizadas en el artículo se centran en analizar el comportamiento del algoritmo en relación con el grafo completo o a la eficacia en detectar los nodos no interesantes. Por ello, esta nueva fase de la experimentación trata de comparar la solución del algoritmo, con lo que podríamos considerar la solución óptima, es decir, la comunidad de la que se extraen los nodos de Q. Todas las pruebas comienzan obteniendo n nodos de una misma comunidad, una comunidad conocida, por lo que se van a comparar los resultados directamente con dicha comunidad.

En este caso, la manera en la que se seleccionan los n vértices de la misma comunidad es algo distinta. El procedimiento es el siguiente:

1. Crear un conjunto de vértices seleccionables S, inicialmente formado por todos los vértices de la comunidad
2. Seleccionar un vértice aleatorio $v \in S$

3. Eliminar de S el vértice v y sus vecinos dentro de la comunidad
4. Si S es vacío terminar con los vértices seleccionados, sino, volver al paso 2

De esta forma, se asegura que los vértices no tienen ninguna conexión entre sí. Además, el número de vértices seleccionados dependerá del tamaño de la comunidad.

A estos n vértices se añaden $m \in \{1, 2, 3, 4\}$ nodos de fuera de la comunidad que funcionarán como valores atípicos, siendo $k = m$, de manera que siempre se tiene la posibilidad de devolver la comunidad original.

Es importante tener en cuenta que con esta técnica es muy posible que, en proporción, el conjunto Q tenga muchos menos nodos que la comunidad que se busca. Incluso, puede llegar a haber más nodos atípicos dentro de Q que de la propia comunidad. Esto puede hacer que los resultados empeoren con respecto a las pruebas anteriores, en las que se tenía un número razonable de nodos de la comunidad a la hora de comenzar el algoritmo.

Las medidas que se van a tomar para comprobar la efectividad del algoritmo en esta fase son la precisión y el recall, explicados anteriormente en la tabla 4. Esta vez, la comparación se realizará entre la comunidad buscada y la solución propuesta por el algoritmo.

Como se puede apreciar, salta a la vista que los resultados para el grafo de Youtube no son para nada buenos con respecto a la precisión para ninguno de los dos algoritmos, mientras que el resto de gráficas parecen representar un buen funcionamiento del algoritmo.

El principal motivo de un resultado tan malo en la precisión es la gran cantidad de falsos positivos. Es decir, el algoritmo identifica con efectividad los nodos pertenecientes a la comunidad, pero, en ese proceso se añaden una gran cantidad de nodos adicionales. Estos nodos son proporcionalmente mucho mayores al tamaño de la propia comunidad, por lo que estropea en gran medida la precisión.

Dado que muchas de las comunidades pueden ser muy cercanas entre sí, es importante analizar estos nodos adicionales que, aunque no forman parte de la comunidad que se busca, pueden ser informativos de cierta manera.

Sin embargo, que la medida del recall sea tan alta nos da la posibilidad de combinar algoritmos, es decir, utilizar este algoritmo para reducir el grafo sin descartar la comunidad para después aplicar otro algoritmo quizá menos eficiente. Esto se debe a que el algoritmo devuelve un grafo de tamaño reducido en el que se encuentra la comunidad buscada. En casos como el de *Amazon* puede contener un pequeño número de nodos adicionales, pero aunque supere en gran cantidad al resultado óptimo siempre va a contener la comunidad buscada dentro del subgrafo generado.

Además, comparando directamente los dos algoritmos genéticos, se puede apreciar una mejoría considerable entre el algoritmo del problema 2 con respecto al del problema 1. Al tener, en general, mejores resultados tanto el precisión como en recall en ambas redes.

Por otra parte, una medida que determina con bastante precisión la calidad de los resultados es la comparación del diámetro de la solución con el diámetro real de la comunidad.

Problema 1		
	amazon	youtube
md3	1.244	2.26
md4	1.051	1.90

Problema 2		
	amazon	youtube
md1	0.932	1.02
md2	0.966	1.11

TABLA 5: Media de los diámetros de las soluciones obtenidas por el algoritmo entre los diámetros de las comunidades que intentaban encontrar para los problemas 1 y 2.

Para el problema 1, todos los resultados respectivos al grafo de Amazon parecen apuntar a una misma dirección, que la comunidad se encuentra casi a la perfección. Además de tener una precisión y un recall alto, los resultados tienden a tener un diámetro ligeramente mayor que el de la comunidad original 5.

Por otra parte, cuando se intentan encontrar las comunidades en el grafo de Youtube, se ha visto que se tiene una precisión bastante menor. Esto se ve reflejado tanto en la precisión como en el diámetro. Aun así, el diámetro no llega al doble del original, que es la precisión del algoritmo original.

El problema 2 obtiene resultados similares en el caso de Amazon, aunque algo mejores con respecto al diámetro. Esto se debe a que, al intentar rechazar el máximo posible de nodos para minimizar el diámetro, deja fuera a algunos de los nodos de la comunidad, creando subcomunidades más pequeñas.

En cambio, los resultados son algo más llamativos en cuanto a la diferencia de diámetros 5 para el grafo de Youtube. Pese a tener una mala precisión, la diferencia de diámetro es mínima. Esto, sumado a que la medida del recall obtiene valores cercanos al 1 indica que realmente se está seleccionando una comunidad que, aunque tiene más nodos de los que debería, sigue siendo realmente compacta.

Las conclusiones que se pueden sacar de este último caso son que las comunidades con las que se trabajan no son completamente precisas, sino que también se pueden encontrar otras comunidades más grandes, pero con la misma conectividad entre sus nodos.

5.6. Tiempos de ejecución

Por último, se van a analizar los tiempos de ejecución para cada una de los casos. Este proyecto ha sido desarrollado en el lenguaje de programación Python, un lenguaje muy utilizado en análisis de datos, pero uno de los más lentos. Por ello, no es coherente hacer una comparación con respecto a los tiempos del artículo. Para comparar los tiempos de ejecución de ambos algoritmos hay que comparar las complejidades temporales de cada uno.

- Problema 1

Para el caso general, se expone que el coste temporal del algoritmo ORIGINAL-P1 es: $\mathcal{O}(|V_G| \cdot (|V_G| + |E_G|))$. Mientras que la complejidad del algoritmo genético para el problema 1 es: $\mathcal{O}(\sqrt[4]{|V_G|} \cdot (|V_G| + |E_G|))$.

Además, en el artículo [1] se expone que el tiempo de ejecución aumenta proporcionalmente al tamaño de k, mientras que el algoritmo genético no sufre ninguna modificación temporal modificando k. Esto puede parecer contradictorio dado que en su complejidad no aparece k, aun así, se puede apreciar cómo el tiempo de ejecución es claramente afectado por el tamaño de k.

Otra ventaja de este algoritmo es que, en su paso inicial, el grafo con el que se trabaja es reducido por su restricción del mínimo grado. Esto hace que tanto el número de vértices como el número de aristas, por tanto el tiempo de ejecución, disminuya con respecto al crecimiento de la restricción del mínimo grado.

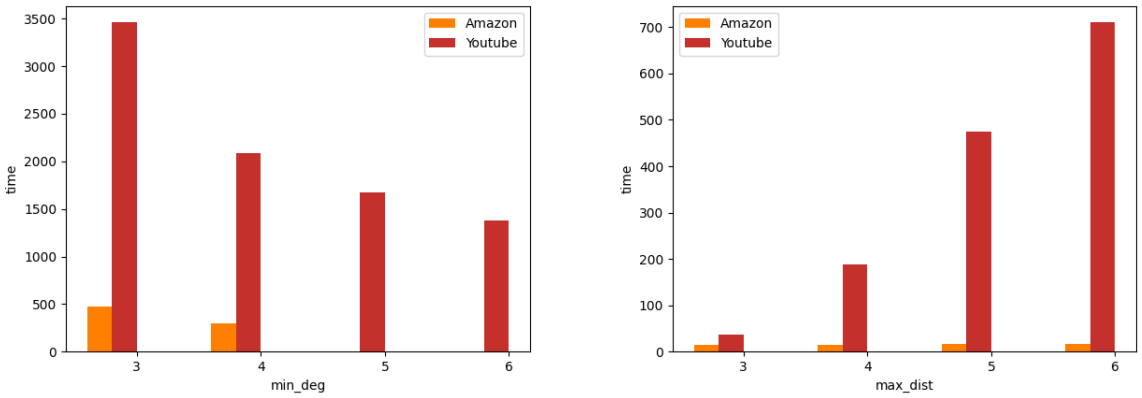
- Problema 2

La complejidad del algoritmo original extraído del artículo es: $\mathcal{O}(|Q| \cdot (|V_H + E_H|) \cdot |V_H|)$. En cambio, la complejidad del algoritmo genético desarrollado es: $\mathcal{O}(|V_G|^{1,25} \cdot |Q|)$.

Al igual que en el caso anterior, el tiempo de ejecución en el algoritmo original crece con respecto de k, a diferencia del genético, en el que el tiempo de ejecución es independiente a k.

En ambos algoritmos el tiempo de ejecución es directamente proporcional a la restricción $dist_{max}$, por lo que los tiempos reales distan mucho de los tiempos teóricos marcados por la complejidad.

Comparando directamente ambos algoritmos genéticos, sí es coherente comparar tiempos de ejecución. Dado que ambos algoritmos han obtenido resultados muy similares en las fases anteriores, el tiempo de ejecución puede desequilibrar la balanza a la hora de determinar qué algoritmo es mejor.



(a) Problema 1

(b) Problema 2

Figura 15: Gráficas que muestran los diferentes tiempos de ejecución dependiendo del algoritmo, en segundos.

Como se puede apreciar en 15 la diferencia de tiempos de ejecución es enorme. A pesar de obtener resultados similares, el algoritmo propuesto para el problema 2 se ejecuta hasta 30 veces más rápido.

5.7. Paralelización

Al trabajar con una gran cantidad de individuos, a los que habitualmente hay que aplicar procesos complejos, se ha optado por palalelizar todos los procesos que tienen que ver con cálculos internos de cada individuo. Esto hace que, dependiendo del número de hilos que se tengan a disposición, se pueda trabajar con poblaciones mayores sin repercutir en el tiempo de ejecución.

En todos los casos relativos a la experimentación se ha trabajado con tamaños de población del doble de los hilos disponibles, por lo que los tiempos de ejecución mostrados podrían reducirse teniendo un mejor equipo.

6. Conclusiones

Para concluir este trabajo vamos a hacer un repaso de cada uno de los algoritmos con los que hemos trabajado. Además, vamos a hacer una comparación de los resultados que nos determine cuál de todos los algoritmos es el que tiene mejor rendimiento.

El objetivo principal era proporcionar una alternativa a los algoritmos de búsqueda de comunidades con valores atípicos explicados en [1] utilizando algortimos genéticos. De los tres problemas que se explican en el artículo hemos seleccionado dos de ellos.

El primero de ellos trata de maximizar el diámetro de la solución utilizando el mínimo grado como restricción. Partiendo de esta base, el algoritmo genético desarrollado utiliza el diámetro como principal medida para el fitness de cada individuo. La restricción se aplica en dos partes, al inicio se modifica el grafo para eliminar los nodos que nunca podrían cumplir dicha restricción, y durante las ejecuciones se modifican los individuos forzándoles a que su mínimo grado sea el adecuado.

Para el segundo problema había dos alternativas, la primera trataba el problema utilizando las mismas medidas que el primer problema pero invirtiendo su uso. Con el fin de desarrollar un segundo algoritmo algo distinto al primero, y, además, eliminando la medida del diámetro, cuya complejidad de cálculo es demasiado grande para un algoritmo heurístico, optamos por desarrollar una segunda versión del problema.

En esta nueva versión se cambia el diámetro como restricción por la distancia desde los nodos de Q del grafo al resto de nodos. Esto nos permite realizar el mismo proceso que en el primer problema, primero se reduce el grafo para no seleccionar nodos que en ningún caso cumplirían la restricción, y después durante las iteraciones se modifican los individuos para adecuarles a la restricción.

Ahora vamos a hacer una lista de cada una de las pruebas de la experimentación, señalando qué algoritmo destaca por encima del resto en cada prueba.

- Detección de valores atípicos

Ambos algoritmos genéticos tienen prácticamente un 100 % de efectividad, mientras que el resto de algoritmos rondan, como mucho, el 90 %.

- Clasificación de outliers

Los resultados obtenidos por ambos algoritmos genéticos son similares a los obtenidos por los algoritmos ORIGINAL-P2.1 y ORIGINAL-P2.2. En esta fase el único algoritmo que se queda atrás es el ORIGINAL-P1.

- Soluciones variando restricciones

Contando con que la medida más significativa de estos resultados es la relación entre el número de nodos del resultado y el número de nodos del grafo original se puede

decir que ambos algortimos genéticos mejoran los resultados del resto de algoritmos. El algoritmo ORIGINAL-P2.2 es el único que obtiene resultados similares.

- Comparación con el óptimo

El algoritmo que más precisión tiene a la hora de buscar la comunidad es claramente el algortimo genético respectivo al problema 2, el cual sin rechazar nodos de la comunidad obtiene subgrafos con prácticamente el mismo diámetro que la comunidad original.

- Tiempo de ejecución

Comparando por complejidad temporal los algoritmos genéticos ganan a los originales. Después, entre ambos el más rápido con diferencia es el respectivo al problema 2.

En conclusión, al tener un rendimiento máximo en cada una de las pruebas el mejor algoritmo para la búsqueda de comunidades de todos los analizados en este trabajo es el algoritmo genético desarrollado para el problema 2.

7. Trabajos futuros

Aunque los resultados han sido prometedores, existen diferentes vías por las que continuar el trabajo y expandir la investigación de búsqueda de comunidades con valores atípicos. A continuación, se exponen diferentes aspectos con los que se podría continuar este trabajo.

7.1. Aumentar coste temporal para intentar aumentar la precisión

En este trabajo, a pesar de obtener buenos resultados, se han utilizado técnicas de aproximación con un factor relativamente malo. Una dirección que se podría tomar en el futuro es completar la experimentación para el problema 1 con la aproximación de factor 1,5 expuesta en 3.1. Esto aumentaría el coste temporal, haciéndolo peor que el del artículo de referencia, pero los resultados podrían mejorar considerablemente. Es algo que no se ha podido llevar a cabo dada la gran cantidad de tiempo que habría tomado la experimentación.

7.2. Optimización de tiempo de ejecución

Una parte negativa de este algoritmo es su implementación. Esto se debe a que ha sido escrito en Python, un lenguaje de programación extremadamente lento comparado con sus rivales. Aplicar el algoritmo en lenguaje C sería un paso por dar muy determinante en términos de usabilidad, ya que se ahorraría mucho tiempo de ejecución.

7.3. Análisis de diferentes parámetros

Los parámetros utilizados como restricción y fitness fueron los utilizados por el artículo original. Un trabajo futuro importante sería explorar varios parámetros que podrían ser utilizados en ambos casos, que podrían mejorar significativamente los resultados. Este análisis podría ajustar más el límite de los algoritmos genéticos en busqueda de comunidades con valores atípicos.

8. Bibliografía

Referencias

[1] Francesco Bonchi, Lorenzo Severini, and Mauro Sozio. Better fewer but better: community search with outliers. In *2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pages 105–112. IEEE, 2020.

[2] Mina Dalirrooyfard, Ray Li, and Virginia Vassilevska Williams. Hardness of approximate diameter: Now for undirected graphs. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1021–1032. IEEE, 2022.

[3] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. Approximate closest community search in networks. *arXiv preprint arXiv:1505.05956*, 2015.

[4] Ángel Kuri, José Galaviz, et al. *Algoritmos genéticos*. Number Sirsi) i9789681663834. IPN, 2002.

[5] Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzone, and Francesco Bonchi. A survey on the densest subgraph problem and its variants. *arXiv preprint arXiv:2303.14467*, 2023.

[6] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 515–524, 2013.

[7] Natali Ruchansky, Francesco Bonchi, David García-Soriano, Francesco Gullo, and Nicolas Kourtellis. To be connected, or not to be connected: That is the minimum inefficiency subgraph problem. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 879–888, 2017.

[8] Mauro Sozio and Aristides Gionis. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 939–948, 2010.