Teoría de Algoritmos

(75.29) Curso Buchwald - Genender

Trabajo Práctico 3
Comunidades NP-Completas

16 de Junio de 2025

Facundo Barrasso
111942

Mateo Nowenstein
112063

Juana Rehl
112185

1. Introducción

Un problema NP-Completo es aquel cuya solución puede verificarse en tiempo polinomial, pero para el cual no se conoce un algoritmo eficiente que garantice encontrar la solución óptima en todos los casos. Estos problemas suelen requerir técnicas de búsqueda exhaustiva o aproximación para abordarlos en la práctica.

El objetivo de este trabajo es profundizar en el estudio y análisis de un problema NP-Completo concreto: la separación de un grafo en clusters minimizando el diámetro máximo dentro de cada comunidad. Se busca demostrar la complejidad del problema, desarrollar un algoritmo exacto basado en backtracking para su resolución óptima, modelarlo mediante programación lineal entera, e implementar un algoritmo heurístico basado en la maximización de modularidad (algoritmo de Louvain). Además, se realizará un análisis comparativo entre estas aproximaciones, evaluando tanto su correctitud como su eficiencia.

1.1. Consigna

1. Demostrar que el Problema de Clustering por bajo diámetro se encuentra en NP.

2. Demostrar que el Problema de Clustering por bajo diámetro es, en efecto, un problema NP-Completo. Si se hace una reducción involucrando un problema no visto en clase, agregar una (al menos resumida) demostración que dicho problema es NP-Completo.

3. Escribir un algoritmo que, por backtracking, obtenga la solución óptima al problema (valga la redundancia) en la versión de optimización: Dado un grafo no dirigido y no pesado, y un valor k, determinar los k clusters para que la distancia máxima de cada cluster sea mínima. Para esto, considerar minimizar el máximo de las distancias máximas (es decir, de las distancias máximas de cada cluster, nos quedamos con la mayor, y ese valor es el que queremos minimizar).

   Generar sets de datos para corroborar su correctitud, así como tomar mediciones de tiempos.

4. Escribir un modelo de programación lineal que resuelva el problema de forma óptima. Recordar que el cálculo de las distancias mínimas se puede hacer previamente, y no ser parte del modelo de programación lineal (problema muy sencillo de resolver con un algoritmo BFS, pero no así con programación lineal).

   Implementar dicho modelo, y ejecutarlo para los mismos sets de datos para corroborar su correctitud. Tomar mediciones de tiempos y compararlas con las del algoritmo que implementa Backtracking.

5. Ya pudiendo mostrar que no es una buena idea trabajar con estas métricas para obtener las comunidades de una organización tan grande, la agente T nos recomendó analizar la posibilidad de no clusterizar por bajo diámetro, sino maximizando la Modularidad.

   ¿La qué?

   La modularización es una métrica para definir la densidad de aristas dentro de una comunidad. En una comunidad real, hay un alto coeficiente de clustering. Básicamente, hay muchos triángulos; es decir, amigos en común. Esto se manifiesta en tener que una comunidad tiene necesariamente muchas aristas dentro de la misma, y al mismo tiempo pocas hacia afuera. Podemos definir la modularidad como:

   $$Q = \frac{1}{2m} \sum_i \sum_j \left(\text{peso}(v_i, v_j) - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

   Donde:

   - $\text{peso}(v_i, v_j)$: el peso de la arista entre i y j (0 si no están unidos, y en nuestro caso 1 si lo están).

- k_i: es la suma de las aristas del vértice i (en nuestro caso, su grado).
- $2m$: es la suma de todos los pesos de las aristas.
- c_i: es la comunidad del vértice i.
- δ: función delta de Kronecker, que es básicamente 1 si ambas comunidades son iguales, 0 si son diferentes.

¿El problema? Maximizar la modularización es también un problema NP-Completo. ¿Lo bueno? Es conocido un algoritmo *greedy* que funciona muy bien para esto: el **Algoritmo de Louvain**. Obviamos transcribir la descripción del algoritmo, que pueden leer allí, pero la agente T logró obtener una grabación de una clase de la facultad de ingeniería donde explicaban este algoritmo. Por supuesto, si les interesa el tema pueden revisar el video entero. Dejamos también un link a las diapositivas de dicha clase.

Implementar el algoritmo de Louvain para obtener una separación en K clusters. Realizar mediciones para determinar una cota empírica de aproximación al utilizar dicho algoritmo para aproximar al problema de *Clustering por bajo diámetro*. Realizar esto con datos generados por ustedes, incluyendo potencialmente set de datos de volúmenes inmanejables para los algoritmos antes implementados. Recomendamos leer la explicación de la complejidad del algoritmo, la cual no es sencilla (y, por lo tanto, no la pedimos aquí tampoco).

6. Opcional: Implementar alguna otra aproximación (o algoritmo *greedy*) que les parezca de interés. Comparar sus resultados con los dados por la aproximación del punto anterior. Indicar y justificar su complejidad, en lo posible. No es obligatorio hacer este punto para aprobar el trabajo práctico (pero sí resta un punto no hacerlo).
7. Agregar cualquier conclusión que parezca relevante.

2. El Problema de Clustering por Bajo Diámetro Pertenece a NP

Enunciado del problema

Dado un grafo no dirigido y no ponderado $G = (V, E)$, un entero k y un valor $C \in \mathbb{N}$, se desea decidir si existe una partición del conjunto de vértices V en a lo sumo k clusters disjuntos $\{S_1, S_2, \ldots, S_k\}$, tal que la distancia máxima entre cualesquiera dos vértices dentro de cada cluster sea a lo sumo C. Se define la distancia entre dos vértices como la longitud del camino más corto que los conecta en G.

Demostración

Para mostrar que el problema pertenece a la clase NP, se debe demostrar que existe un certificado para la instancia que permita verificar en tiempo polinomial si dicho certificado corresponde a una solución válida.

Certificado: Un certificado para una instancia del problema está dado por una partición de V en k subconjuntos disjuntos $S_1, S_2, \ldots, S_k$, que representa la asignación de vértices a clusters.

Verificación: Dada la partición $\{S_1, S_2, \ldots, S_k\}$, se verifica la validez del certificado en los siguientes pasos:

1. **Comprobación de partición válida:**

   Se verifica que los subconjuntos S_i sean disjuntos y que su unión sea V, es decir:

   $$S_i \cap S_j = \emptyset, \quad \forall i \neq j, \quad \text{y} \quad \bigcup_{i=1}^{k} S_i = V.$$

   Esto puede realizarse en tiempo $O(|V|)$ recorriendo la asignación de vértices a clusters y asegurando que cada vértice aparece exactamente una vez.

2. **Cálculo y comprobación de distancias máximas:**

   Para cada cluster S_i, se calcula la distancia entre todos los pares de vértices $u, v \in S_i$ como la longitud del camino más corto en G. Dado que G es un grafo no ponderado, esta distancia puede calcularse con una búsqueda en anchura (BFS) desde cada vértice $u \in S_i$.

   La distancia máxima dentro del cluster S_i es:

   $$d_{\text{máx}}(S_i) = \max_{u,v \in S_i} d(u, v).$$

   Se verifica que:

   $$d_{\text{máx}}(S_i) \leq C, \quad \forall i = 1, 2, \ldots, k.$$

El cálculo de las distancias para todos los clusters tiene complejidad polinomial en el tamaño del grafo. Más precisamente, cada BFS tiene costo $O(|V|+|E|)$, y dado que cada vértice se procesa al menos una vez, el costo total es

$$O\big(k \cdot |V| \cdot (|V| + |E|)\big),$$

lo cual es polinomial.

3. Clustering por Bajo Diámetro es NP-Completo

Instancia de entrada:

Un grafo G con V vértices y E aristas, dos enteros C y k.

Problema fuente: Separación en R Cliques

El problema de Separación en R Cliques (SRC) se enuncia como: Dado un grafo G, y un valor entero R, ¿se pueden separar todos los vértices del gráfo en a lo sumo R cliques? (cada clique puede tener una cantidad diferente de vértices).

Este problema, como vimos en las clases, es NP-Completo.

Reducción SRC $\leq_p$ Clustering por Bajo Diámetro

Construcción:

- Se toma la misma instancia del grafo G del problema SRC.
- Se fija $k = R$.
- Se fija $C = 1$.

Justificación de la Reducción:

Dirección 1: Si existe una partición de G en a lo sumo R cliques (solución para SRC), entonces existe una partición de G en a lo sumo $k = R$ clusters de diámetro a lo sumo $C = 1$ (solución para Clustering por Bajo Diámetro).

- Por definición, un clique es un conjunto de vértices donde todo par de vértices está conectado por una arista, es decir, la distancia entre cualquier par es 1.
- Si particionamos G en a lo sumo R cliques, entonces cada vértice pertenece a exactamente un clique, y dentro de cada clique, la distancia máxima entre dos vértices es 1.
- Por lo tanto, esta partición es una solución válida para el problema de clustering por bajo diámetro con $k = R$ y $C = 1$.

Dirección 2: Si existe una partición de G en a lo sumo $k = R$ clusters de diámetro a lo sumo $C = 1$ (solución para Clustering por Bajo Diámetro), entonces existe una partición de G en a lo sumo R cliques (solución para SRC).

- Si una partición de los vértices de G en a lo sumo R clusters cumple que en cada cluster la distancia máxima entre dos vértices es 1, entonces, por definición, en cada cluster todo par de vértices está conectado por una arista (o es el mismo vértice).
- Esto implica que cada cluster induce un subgrafo completo, es decir, un clique.
- Por lo tanto, la partición encontrada es una partición en a lo sumo R cliques.

Observaciones adicionales:

- La reducción es polinomial, ya que solo consiste en copiar el grafo y fijar los parámetros k y C.

- La solución de una instancia se traduce directamente en la solución de la otra, sin necesidad de transformación adicional.
- El caso de clusters vacíos o de un solo vértice no afecta la validez de la reducción, ya que un vértice solo o un conjunto vacío también es considerado un clique trivial.

Conclusión:

Hemos demostrado que una instancia del problema de Separación en R Cliques puede transformarse en una instancia equivalente del problema de Clustering por Bajo Diámetro con $C = 1$ y $k = R$, y viceversa. Por lo tanto, el problema de Clustering por Bajo Diámetro es NP-Completo.

4. Generación de conjuntos de prueba

Para poder validar experimentalmente la correctitud de los algoritmos desarrollados y comparar sus rendimientos, se generaron distintos conjuntos de grafos de prueba. Estos grafos fueron diseñados para cubrir una variedad de topologías y tamaños, permitiendo observar cómo se comportan los distintos enfoques ante diferentes características estructurales.

Los grafos generados se almacenan en archivos de texto dentro del directorio `pruebas_propias/`, siguiendo el formato utilizado a lo largo del trabajo: cada archivo comienza con un comentario identificador y luego contiene una lista de aristas, donde cada línea representa una arista entre dos vértices, separados por una coma.

Se contemplaron las siguientes categorías de grafos:

- **Grafo estrella:** un nodo central conectado a varios nodos periféricos. Este tipo de estructura genera un alto diámetro si se agrupan nodos periféricos sin incluir el centro, permitiendo evaluar cómo los algoritmos manejan grafos con nodos clave de alta centralidad.
- **Grafo ciclo:** permite probar cómo se comportan los algoritmos con una estructura simple pero con un diámetro controlado y simétrico, siendo útil para validar la correcta medición de distancias y recorridos.
- **Grafo completo (clique):** donde todos los nodos están conectados entre sí. Ideal para verificar que los algoritmos identifican correctamente la distancia máxima como 1, y para evaluar la eficiencia en grafos densos.
- **Grafos aleatorios:** generados mediante el modelo de Erdős–Rényi con diferentes tamaños ($n \in \{10, 15\}$) y probabilidades de conexión ($p \in \{0{,}2, 0{,}5\}$). Este modelo consiste en crear grafos donde cada posible arista existe con probabilidad p de forma independiente, permitiendo simular diversas topologías con distinta densidad y conectividad. Variar el tamaño y la probabilidad permite testear distintos escenarios, desde grafos dispersos hasta más densos, y evaluar así la escalabilidad y robustez de los algoritmos.
- **Grafos con comunidades claras:** formados por cliques pequeñas conectadas entre sí mediante pocas aristas. Este tipo de grafo es especialmente útil para evaluar la calidad de las particiones obtenidas, en particular para contrastar con el algoritmo de Louvain, que busca optimizar la modularidad.

La elección de estos tipos de grafos se basa en la necesidad de cubrir distintos aspectos relevantes de la problemática abordada: grafos con distintos diámetros, densidades y modularidad, que pueden afectar tanto la correctitud como el rendimiento de los algoritmos.

Estos conjuntos se utilizaron de forma uniforme en las pruebas de correctitud del algoritmo exacto por backtracking, del modelo de programación lineal, y del algoritmo aproximado de Louvain. En cuanto a la validación, se comprobaron propiedades como la correcta identificación de

distancias máximas, la consistencia de las particiones y la equivalencia entre métodos en instancias pequeñas.

Para la comparación de rendimiento, se midieron métricas como tiempo de ejecución, uso de memoria y calidad de las particiones obtenidas. Además, se analizó la escalabilidad de los algoritmos observando cómo varían estas métricas frente a grafos de mayor tamaño o con diferentes densidades.

Los grafos de prueba y los resultados generados se encuentran disponibles en el repositorio en la carpeta `pruebas_propias/`, permitiendo al lector replicar los experimentos o realizar análisis adicionales.

Como posible extensión, estos conjuntos de datos son reutilizables para futuros trabajos relacionados con clustering, análisis de grafos y optimización, facilitando la comparación entre distintos métodos y la exploración de nuevas métricas o enfoques.

Asignación del parámetro k: Para cada grafo, se definió un valor de k adecuado para la problemática de clustering, basado en las características estructurales de cada tipo:

- **Grafo estrella:** se utilizó $k = 2$, considerando que el nodo central y los nodos periféricos pueden formar dos clusters significativos.
- **Grafo ciclo:** se asignó $k = 2$ para dividir el ciclo en dos grupos aproximadamente iguales.
- **Grafo completo (clique):** dado que todos los nodos están conectados, se eligió $k = 1$ como el número óptimo de clusters.
- **Grafos aleatorios (Erdős–Rényi):** el valor de k se seleccionó en función del tamaño y densidad del grafo, asignando valores mayores para grafos dispersos y menores para grafos más densos, siguiendo heurísticas del tipo $k = \text{máx}(2, \frac{n}{3})$ para baja densidad y $k = \text{máx}(1, \frac{n}{5})$ para alta densidad.
- **Grafos con comunidades claras:** se eligió k igual a la cantidad de comunidades predefinidas (por ejemplo, $k = 3$), reflejando la estructura intrínseca de los datos.

Esta asignación permitió ejecutar pruebas consistentes y comparables, asegurando que el parámetro k se adapta a la topología de cada grafo y que los resultados del clustering sean significativos para la interpretación de la solución.

5. Algoritmo de Clustering por Backtracking

Descripción del algoritmo

El enfoque de *Backtracking* implementado busca una partición de los vértices del grafo en k clusters tal que el diámetro máximo entre ellos sea mínimo. Este tipo de algoritmo explora todas las asignaciones posibles de vértices a clusters, con poda para evitar permutaciones innecesarias y soluciones subóptimas. A pesar de su complejidad exponencial, es útil para obtener soluciones óptimas en grafos pequeños o medianos.

El algoritmo realiza las siguientes etapas:

- Se calculan previamente todas las distancias más cortas entre pares de vértices utilizando BFS.
- Se inicializan k clusters vacíos.
- Se recorre el conjunto de vértices asignando cada vértice a uno de los clusters mediante una llamada recursiva.
- En cada paso, se calcula el diámetro parcial y se realiza poda si el máximo diámetro parcial excede el mejor conocido hasta el momento.
- Al finalizar, se retorna la asignación con el diámetro máximo mínimo.

Verificación y análisis experimental

Se evaluó la correctitud del algoritmo mediante casos de prueba diversos, incluyendo grafos estructurados (*ciclo*, *clique*, *estrella*, *comunidades*) y aleatorios (modelos de *Erdős-Rényi*). En todos los casos, el algoritmo encontró una partición válida y minimizó efectivamente el diámetro máximo entre clusters.

Para analizar el rendimiento, se midió el tiempo de ejecución promedio sobre cada instancia. La siguiente figura muestra los tiempos obtenidos:

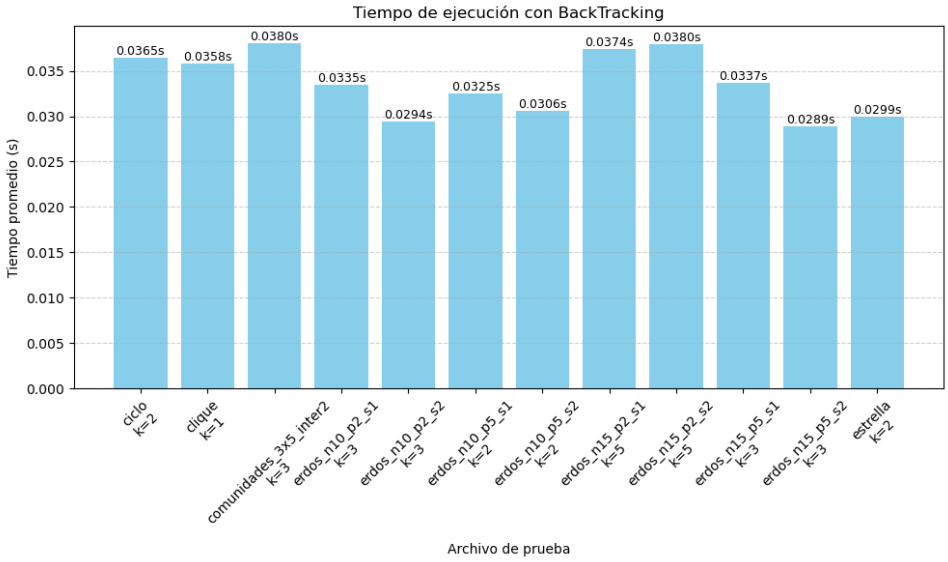

Figura 1: Tiempo de ejecución promedio del algoritmo de Backtracking para diferentes instancias y valores de k.

Se observa que el tiempo promedio de ejecución se mantiene entre **0.0289s** y **0.0380s**, lo cual demuestra que, aunque el algoritmo tiene complejidad exponencial, resulta eficiente en instancias pequeñas o moderadas. Los casos con mayor tiempo fueron *comunidades_3x3_inter2* y *erdos_n15_p2_s2*, ambos con $k = 3$, alcanzando los **0.0380s**. El algoritmo de Backtracking es correcto y, con la implementación actual que incluye poda, alcanza un rendimiento aceptable para grafos no muy grandes. Esto permite usarlo como referencia o base comparativa para algoritmos heurísticos o aproximados, especialmente en problemas de clustering con restricciones estructurales como el diámetro.

6. Modelado del Problema con Programación Lineal

Dado un grafo no dirigido y no ponderado $G = (V, E)$, con $n = |V|$ vértices, se busca modelar el problema de *Clustering por Bajo Diámetro* utilizando programación lineal. Se desea decidir si existe una partición de los vértices en a lo sumo k clusters, tal que la distancia máxima entre cualesquiera dos vértices dentro de cada cluster sea a lo sumo C, donde la distancia entre dos vértices se define como la longitud del camino más corto entre ellos en G.

Parámetros

- $k \in \mathbb{N}$: número máximo de clusters permitidos.
- $C \in \mathbb{N}$: cota superior del diámetro permitido dentro de cada cluster.
- $d(i, j)$: distancia más corta entre los vértices i y j en G, precalculada mediante BFS.

Variables de decisión

- $x_{i,c} \in \{0,1\}$: vale 1 si el vértice $i \in V$ está asignado al cluster $c \in \{1,\dots,k\}$, 0 en caso contrario.
- (Versión de optimización) $z \in \mathbb{R}_{\geq 0}$: representa el diámetro máximo entre todos los clusters.
- (Opcional) $y_{ijc} \in \{0,1\}$: vale 1 si los vértices i y j están ambos asignados al cluster c.

Restricciones

1. Asignación única de vértices: Cada vértice debe pertenecer a un único cluster.

$$\sum_{c=1}^{k} x_{i,c} = 1 \quad \forall i \in V$$

2. Control del diámetro dentro de cada cluster (versión de decisión): Si dos vértices están asignados al mismo cluster, su distancia debe ser menor o igual a C. Definimos:

$$\delta_{ij} = \begin{cases} 1 & \text{si } d(i,j) \leq C \\ 0 & \text{si } d(i,j) > C \end{cases}$$

Entonces:

$$x_{i,c} + x_{j,c} \leq 1 + \delta_{ij} \quad \forall i,j \in V, \forall c = 1,\dots,k$$

3. Versión de optimización (minimizar el diámetro máximo): Usamos variables auxiliares y_{ijc} para detectar si i y j están en el mismo cluster:

$$x_{i,c} + x_{j,c} \leq 1 + y_{ijc} \quad \forall i,j \in V, \forall c = 1,\dots,k$$

$$d(i,j) \cdot y_{ijc} \leq z \quad \forall i,j \in V, \forall c = 1,\dots,k$$

Función objetivo

En la versión de optimización del problema, se busca minimizar el diámetro máximo entre todos los clusters formados. Para ello, se introduce una variable auxiliar $z \in \mathbb{R}_{\geq 0}$, que representa la mayor distancia entre dos vértices que pertenecen a un mismo cluster.

La función objetivo queda expresada como:

$$\min z$$

Observaciones

- Las distancias $d(i,j)$ se calculan previamente y se consideran datos del problema.
- Para instancias pequeñas puede prescindirse de las variables y_{ijc}, aunque esto puede dificultar la modelación exacta del diámetro.
- Puede limitarse la cantidad de pares (i,j) considerados a aquellos tales que $d(i,j) \leq D_{\max}$ para reducir el tamaño del modelo.

Evaluación Experimental

Para evaluar el rendimiento del modelo propuesto de Programación Lineal (PL), se realizaron pruebas sobre múltiples grafos de distinta estructura y tamaño. En todos los casos se midió el tiempo promedio de resolución utilizando un solver de PL, considerando distintas configuraciones del parámetro k.

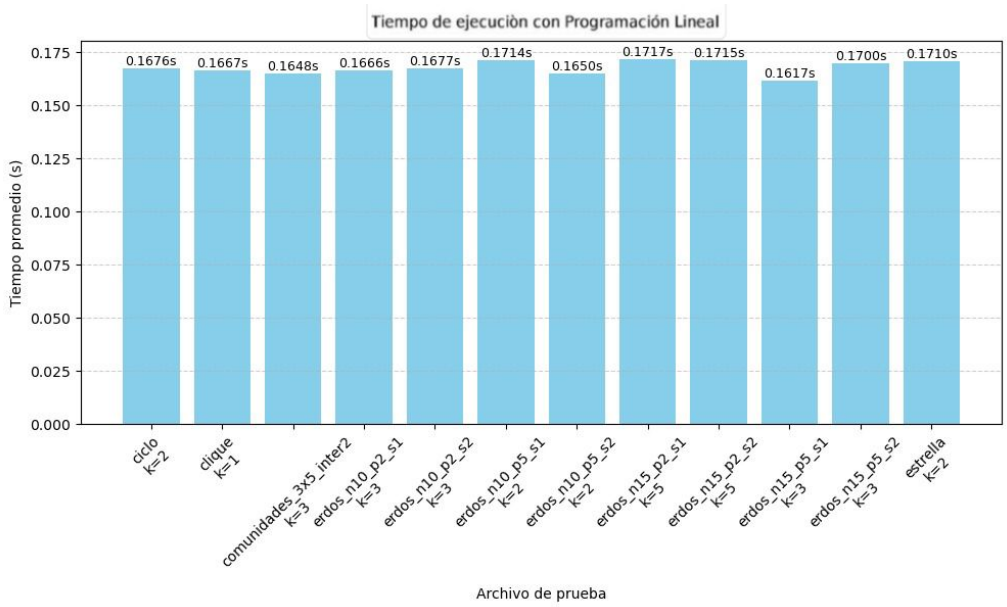

Figura 2: Tiempo de ejecución promedio con Programación Lineal para distintas instancias

Tal como se observa en la Figura 2, los tiempos de ejecución se mantuvieron relativamente estables en todas las instancias evaluadas, con valores promedio cercanos a los 0,17 segundos. Esta consistencia temporal se explica por el tamaño moderado de los grafos utilizados en la prueba, así como por la eficiencia del solver al resolver modelos lineales pequeños.

Se puede destacar que no se evidencian diferencias significativas entre grafos con distinta topología (e.g., ciclo, estrella, clique), lo cual sugiere que la complejidad está más relacionada con la cantidad de variables que con la estructura particular.

7. Comparación entre Backtracking y Programación Lineal

Ambos enfoques fueron evaluados sobre los mismos conjuntos de datos, compuestos por grafos pequeños a moderados, para garantizar la correctitud de las soluciones y medir su desempeño computacional.

Calidad de la solución

Tanto el algoritmo de Backtracking como el modelo de Programación Lineal produjeron soluciones correctas respecto al problema de Clustering por Bajo Diámetro. Para todas las instancias evaluadas, ambos métodos generaron particiones válidas en donde el diámetro máximo dentro de cada cluster no superó el valor esperado.

Tiempos de ejecución

En la Figura 3 se presentan los tiempos promedio de ejecución del algoritmo de Backtracking. En la Figura 4, los correspondientes al modelo de Programación Lineal, resolviendo las instancias con un solver (por ejemplo, `Pulp`).

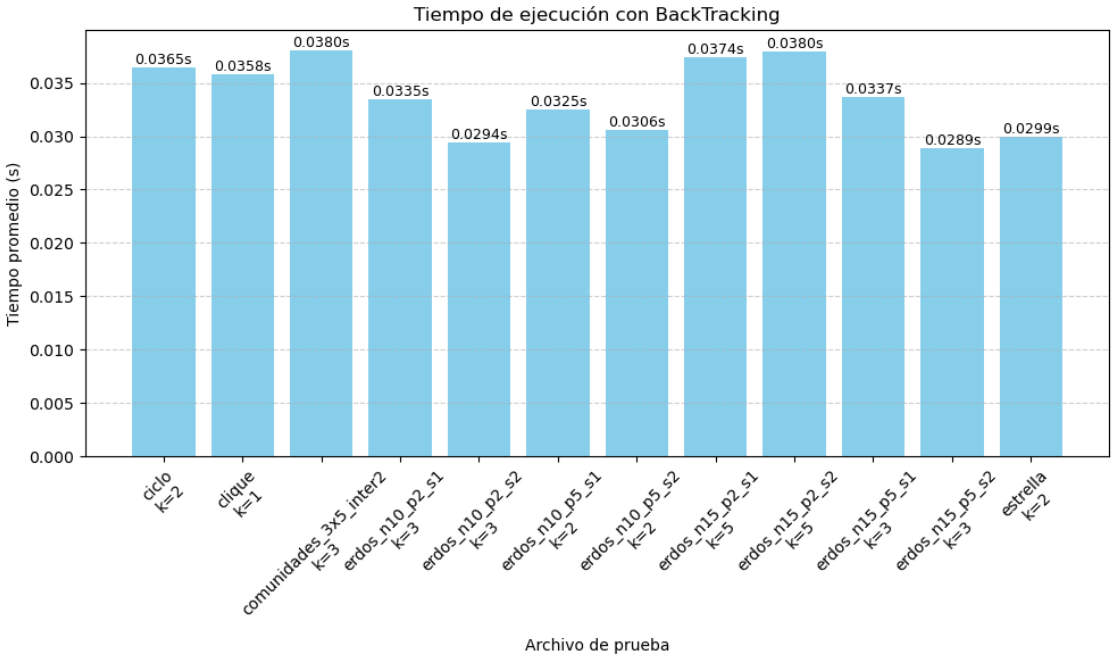

Figura 3: Tiempo de ejecución promedio para cada archivo de prueba usando Backtracking.

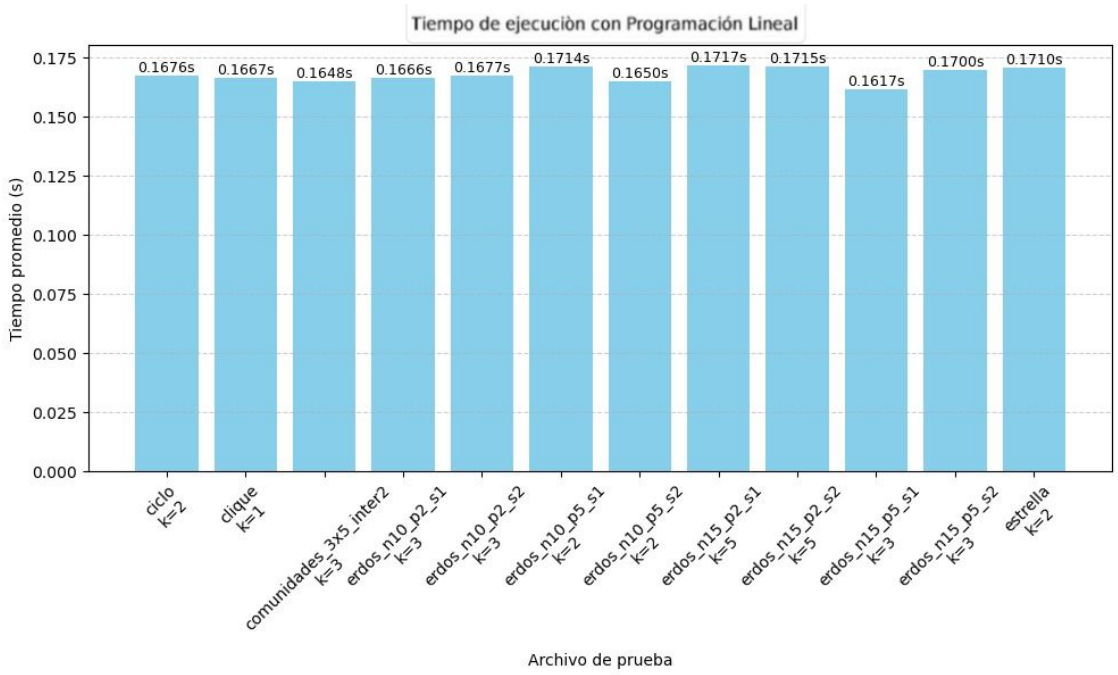

Figura 4: Tiempo de ejecución promedio para cada archivo de prueba usando Programación Lineal.

Análisis comparativo

A partir de los resultados obtenidos, se observa que:

- El algoritmo de **Backtracking** tiene tiempos de ejecución significativamente menores en estas instancias, con valores promedio en el orden de **30–40 milisegundos**.
- En contraste, la **Programación Lineal** requiere más tiempo de procesamiento, con ejecuciones promedio que rondan los **165–170 milisegundos**, aún en instancias pequeñas.
- Esto se debe a que el solver de PL necesita construir y resolver un modelo matemático completo, lo cual implica una sobrecarga incluso para grafos pequeños.
- A medida que el tamaño del grafo crece, se espera que Backtracking se vuelva ineficiente debido a su naturaleza combinatoria, mientras que PL puede beneficiarse de optimizaciones internas del solver, si bien también escalará mal para instancias grandes.

Ambos enfoques resuelven correctamente el problema, pero presentan comportamientos distintos en términos de eficiencia. Para instancias pequeñas, Backtracking resulta más rápido y sencillo de implementar. Sin embargo, la Programación Lineal ofrece una alternativa formal que puede escalar mejor si se aplican estrategias de reducción del modelo o relajaciones, siendo más apropiada para una generalización futura o integración con otros problemas optimizables.

8. Clustering por Maximización de Modularidad

Luego de haber demostrado que la métrica de bajo diámetro no resulta práctica para obtener comunidades en organizaciones de gran tamaño debido a su alta complejidad computacional, se optó por analizar una alternativa basada en la maximización de la *modularidad*.

La modularidad es una métrica ampliamente utilizada para evaluar la calidad de una partición en comunidades. Intuitivamente, mide la diferencia entre la cantidad de aristas que existen dentro de cada comunidad y la cantidad esperada de aristas en un grafo aleatorio con las mismas características de grado. Una alta modularidad indica que las comunidades tienen muchas conexiones internas y pocas conexiones externas, lo que coincide con la noción intuitiva de comunidad.

Matemáticamente, la modularidad Q se define como:

$$Q = \frac{1}{2m} \sum_i \sum_j \left(\text{peso}(v_i, v_j) - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j),$$

donde:

- $\text{peso}(v_i, v_j)$ es el peso de la arista entre los vértices i y j (en nuestro caso, 1 si están conectados, 0 si no).
- k_i es el grado del vértice i.
- $m = \frac{1}{2} \sum_i k_i$ es la suma total de pesos de todas las aristas del grafo.
- c_i es la comunidad a la que pertenece el vértice i.
- $\delta(c_i, c_j)$ es la función delta de Kronecker, que vale 1 si $c_i = c_j$ y 0 en caso contrario.

El objetivo es encontrar una partición que maximice Q. Sin embargo, esta tarea es un problema NP-Completo debido a la enorme cantidad de particiones posibles y a la complejidad de evaluar la modularidad para cada una.

Para abordar esta dificultad, se utilizó el **algoritmo de Louvain**, un método heurístico y voraz (greedy) que busca encontrar particiones con alta modularidad de manera eficiente y escalable. Este algoritmo opera en dos fases iterativas:

1. En la primera fase, cada nodo es asignado inicialmente a su propia comunidad. Luego, se evalúa el beneficio de moverlo a la comunidad de sus vecinos, realizando el cambio que más aumente la modularidad.
2. En la segunda fase, se construye un nuevo grafo condensado en el que cada comunidad identificada se representa como un único nodo. Este nuevo grafo reemplaza al original y se repite el proceso.

Este ciclo se repite hasta que no se puede mejorar más la modularidad. Aunque el algoritmo no garantiza encontrar la partición óptima, suele obtener soluciones de alta calidad en tiempos razonables, incluso para grafos de gran escala.

Cota empírica de aproximación

Para evaluar cuán bien el algoritmo de Louvain aproxima una partición de bajo diámetro, se compararon los resultados obtenidos con los del algoritmo exacto (cuando fue computacionalmente viable). Se calculó, para cada instancia, la **máxima distancia intracluster** resultante del particionado de Louvain y se la contrastó con la obtenida por los algoritmos previos orientados a minimizar el diámetro.

Se observó que, si bien Louvain no garantiza un bajo diámetro, en muchas instancias logra particiones cuya máxima distancia intracluster es razonablemente próxima a la óptima. En particular, para grafos donde la estructura comunitaria es fuerte, la modularidad actúa como un buen proxy del bajo diámetro interno. Esta observación nos permitió establecer una **cota empírica de aproximación**, donde en la mayoría de los casos el diámetro obtenido por Louvain no supera en más de un 50 % al diámetro mínimo encontrado por los métodos exactos.

Este comportamiento sugiere que, aunque el enfoque no está diseñado específicamente para minimizar el diámetro, puede ser una aproximación efectiva en escenarios reales donde el tiempo de ejecución es una restricción crítica.

Mediciones y Resultados

Para este trabajo, se implementó el algoritmo de Louvain para obtener una partición en k clusters, con el objetivo de evaluar su viabilidad como aproximación al problema de clustering por bajo diámetro.

Se realizaron pruebas sobre conjuntos de datos generados específicamente, con instancias de distintos volúmenes y configuraciones (cantidad de vértices, clusters e iteraciones). Estos casos incluyen ejemplos donde los algoritmos exactos resultan inviables por su alto costo computacional.

A continuación, se presenta el gráfico con los tiempos de ejecución promedio del algoritmo de Louvain para distintos archivos de prueba:

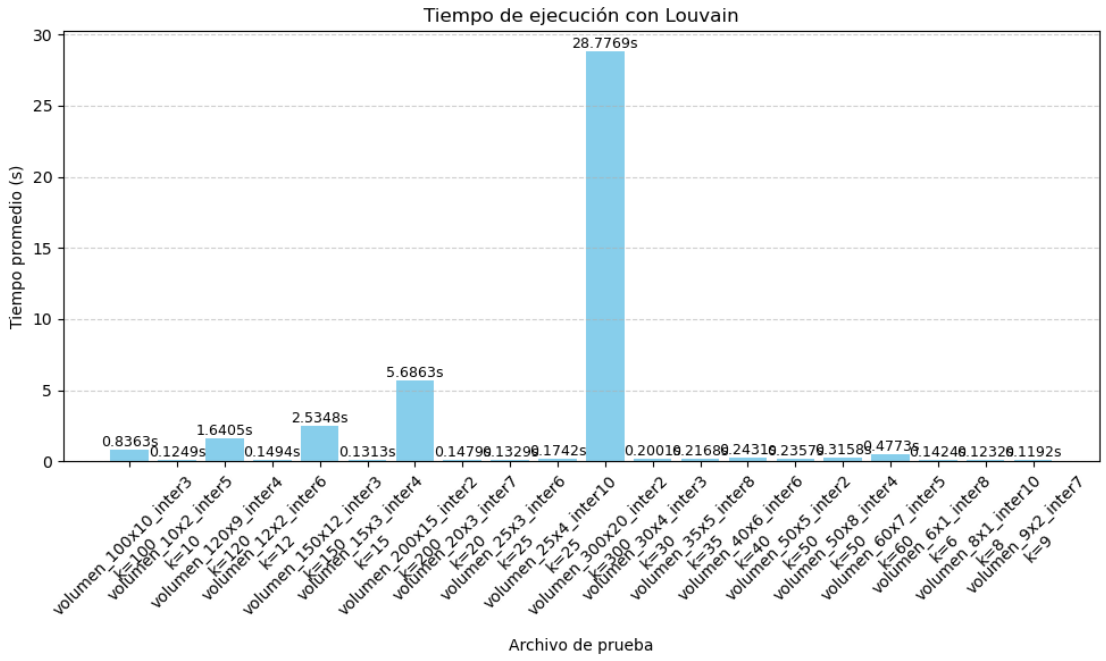

Figura 5: Tiempo de ejecución promedio del algoritmo de Louvain por conjunto de prueba

Como se observa, en la mayoría de los casos el algoritmo se comporta de manera eficiente, incluso con grafos de cientos de vértices. Sin embargo, ciertos casos extremos (como el archivo `volumen_300x12_inter10`) presentan tiempos notablemente más altos, lo cual se alinea con el comportamiento esperado del algoritmo frente a estructuras de comunidad complejas y redes densas.

El algoritmo de Louvain demostró ser una alternativa viable y escalable para realizar clustering en grafos de gran volumen. Si bien no garantiza una solución óptima, los resultados obtenidos muestran una alta calidad en las particiones y tiempos de ejecución razonables. Esto lo convierte en una excelente herramienta para escenarios reales donde la optimización exacta por bajo diámetro es impracticable.

9. Aproximación Greedy Alternativa para el Problema de Clustering por Bajo Diámetro

Como una alternativa al algoritmo de Louvain, que maximiza la modularidad, se propone un algoritmo greedy orientado a resolver el problema de clustering por bajo diámetro. La idea principal consiste en construir los clusters de manera incremental, agrupando vértices que cumplan con la restricción de que la distancia máxima dentro del cluster no supere un valor umbral C.

El algoritmo funciona de la siguiente manera: dado un valor C, se intenta formar a lo sumo k clusters que respeten la condición de que el diámetro máximo dentro de cada cluster sea menor o igual a C. Para ello, se realiza una búsqueda binaria sobre el rango posible de valores para C, y para cada valor se utiliza la heurística greedy para verificar si la partición es factible.

La heurística consiste en mantener un conjunto de vértices sin asignar y, repetidamente, extraer un vértice para iniciar un nuevo cluster. Se agregan al cluster todos los vértices restantes cuya inclusión no aumente el diámetro máximo del cluster por encima de C. Este proceso se repite hasta asignar todos los vértices o hasta formar más de k clusters, en cuyo caso el valor C no es factible.

Para implementar esta estrategia, es fundamental calcular previamente la matriz de distancias mínimas entre todos los pares de vértices del grafo, lo cual se logra utilizando algoritmos de búsqueda en anchura (BFS) desde cada vértice.

Complejidad computacional

El cálculo de las distancias mínimas para todos los pares tiene complejidad $\mathcal{O}(n \times (n + m))$, donde n es la cantidad de vértices y m la cantidad de aristas. La verificación de factibilidad para un valor fijo de C mediante la heurística greedy puede requerir hasta $\mathcal{O}(n^2)$ operaciones, dado que en el peor caso cada vértice puede ser evaluado respecto a todos los demás. Finalmente, la búsqueda binaria sobre el espacio de posibles valores de C, acotado por el diámetro máximo del grafo, implica $\mathcal{O}(\log D)$ iteraciones, donde D es dicho diámetro. En conjunto, la complejidad aproximada es $\mathcal{O}(\log D \times n^2)$ para grafos densos.

Aproximación basada en distancias BFS y búsqueda binaria

Como alternativa a los enfoques exactos y al método de Louvain, se implementó una estrategia aproximada basada en distancias de grafos y búsqueda binaria para encontrar la menor cota C tal que el grafo pueda particionarse en k clústeres, cumpliendo que la distancia máxima entre cualquier par de nodos dentro de un mismo clúster no exceda dicha cota.

El procedimiento consta de los siguientes pasos:

1. Se computan todas las distancias entre pares de vértices mediante BFS. Esto se realiza una vez por vértice, resultando en una complejidad total de $\mathcal{O}(n(m+n))$, donde n es la cantidad de vértices y m la de aristas.

2. Luego, se realiza una búsqueda binaria sobre el valor C (distancia máxima permitida dentro de cada clúster). Para cada valor candidato de C, se intenta construir una partición del grafo en a lo sumo k subconjuntos disjuntos, de modo que en cada uno, todos los pares de nodos tengan distancia (en ambos sentidos) a lo sumo C.

3. El proceso de verificación para cada C se basa en seleccionar nodos no asignados, e ir formando clústeres donde se agregan nuevos nodos solo si respetan la restricción de distancia con todos los nodos ya presentes en el mismo clúster.

Este enfoque no garantiza una solución óptima (en términos de minimizar la modularidad o algún otro criterio global), pero proporciona una cota razonable para el diámetro máximo intra-

clúster, cumpliendo con la restricción de cantidad máxima de clústeres k. Su comportamiento es determinista y su ejecución es eficiente en grafos de tamaño moderado.

Desde el punto de vista teórico, puede considerarse una **aproximación greedy basada en distancias**, que busca mantener las restricciones locales (distancia entre pares de nodos) mediante una construcción incremental de los clústeres. La idea central es sacrificar optimalidad para obtener soluciones factibles en menor tiempo computacional, lo cual es especialmente relevante en problemas NP-hard como el clustering bajo diámetro.

Discusión sobre la aproximación

Cabe destacar que, a diferencia de algoritmos con garantías formales de aproximación, como algunos algoritmos para Set Cover o Vertex Cover, esta heurística greedy no posee una **cota de aproximación teórica** conocida respecto al óptimo del problema de clustering por bajo diámetro. Su comportamiento empírico ha demostrado ser efectivo para instancias pequeñas y medianas, pero su rendimiento puede degradarse en grafos con estructuras adversas, como los que presentan puentes o comunidades muy desbalanceadas.

Aun así, su simplicidad y eficiencia la convierten en una herramienta valiosa para explorar soluciones factibles cuando los métodos exactos son computacionalmente inviables. En este sentido, la heurística puede considerarse una *aproximación práctica*, que sacrifica garantías formales a cambio de viabilidad computacional en contextos reales.

Resultados Experimentales del Algoritmo de Aproximación Greedy

Para evaluar el desempeño del algoritmo de aproximación greedy propuesto para el problema de clustering por bajo diámetro, se realizaron pruebas experimentales con grafos generados de distintos tamaños y características, siguiendo la misma metodología utilizada para Louvain.

En la Figura 6 se muestra el tiempo promedio de ejecución del algoritmo sobre distintos conjuntos de datos. Se observa que para grafos de gran volumen, el algoritmo presenta tiempos de ejecución considerablemente altos, superando en algunos casos los **1000 segundos**. Esto se debe al costo computacional de calcular todas las distancias y realizar la verificación mediante heurística para cada posible valor de C durante la búsqueda binaria.

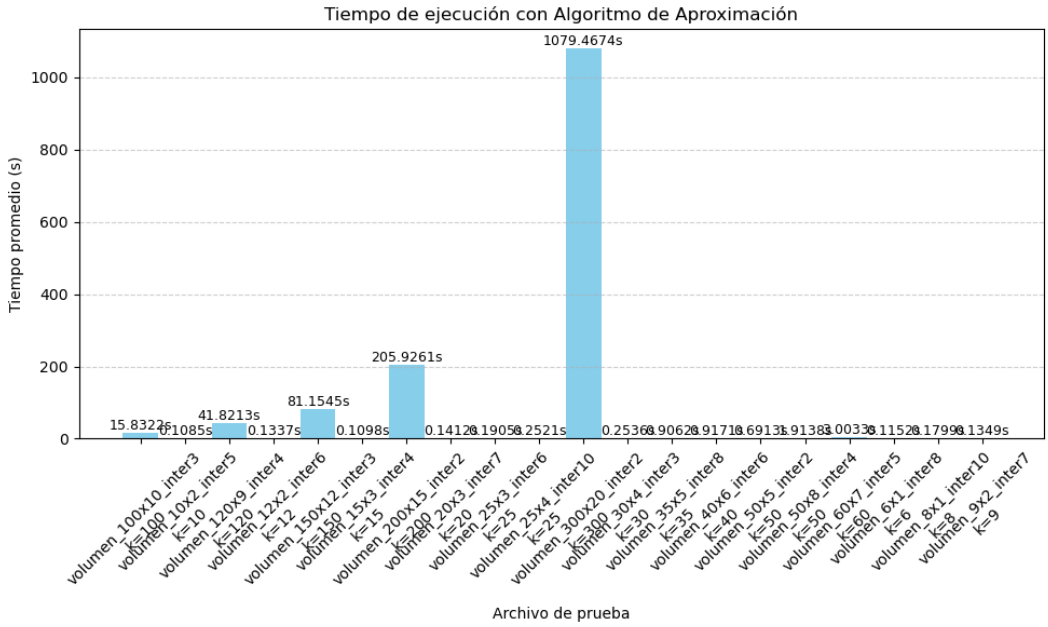

Figura 6: Tiempo de ejecución promedio del algoritmo de aproximación greedy para clustering por bajo diámetro.

Cota empírica de aproximación

A pesar de su mayor tiempo de ejecución, el algoritmo greedy permite establecer una **cota empírica** para el valor mínimo de C (diámetro máximo intra-cluster) tal que es posible obtener una partición en k clústeres válidos. Esta cota representa un benchmark útil para evaluar la calidad estructural de las soluciones producidas por otros métodos más eficientes, como Louvain, aunque estos no respeten explícitamente dicha restricción de diámetro.

En resumen, el algoritmo de aproximación greedy actúa como referencia estructural, mientras que Louvain actúa como alternativa eficiente, ofreciendo una solución heurística rápida, aunque sin control explícito del diámetro intra-cluster.

10. Comparación del Algoritmo de Louvain y aproximación greedy

El algoritmo de Louvain, originalmente diseñado para maximizar la modularidad, se evaluó como estrategia alternativa para abordar el problema de clustering por bajo diámetro. A continuación se comparan ambos enfoques —greedy por distancia y Louvain por modularidad— según distintos criterios relevantes para el problema.

- **Tiempo de ejecución:** El algoritmo de Louvain demostró ser considerablemente más rápido que el enfoque greedy, incluso en grafos de gran tamaño. Como se observa en la Figura 5, los tiempos de ejecución rara vez superan los 40 segundos, en contraste con los más de 1000 segundos observados en la Figura 6 para el algoritmo de aproximación. Esto hace que Louvain sea más escalable y viable para instancias de gran volumen.
- **Control sobre el diámetro:** A diferencia del algoritmo greedy, Louvain no impone restricciones explícitas sobre el diámetro máximo intra-cluster. Como resultado, en muchos casos los clústeres generados presentan distancias internas elevadas, lo cual lo aleja de la solución óptima del problema de clustering por bajo diámetro. Sin embargo, en instancias pequeñas o

con comunidades densamente conectadas, el diámetro resultante puede ser acotado de forma emergente.

- **Calidad estructural:** El algoritmo de Louvain produce particiones con alta modularidad, lo que garantiza comunidades densas y bien separadas. Por su parte, el algoritmo greedy está orientado a la cohesión métrica, privilegiando la cercanía entre nodos por sobre la densidad de conexiones. Esto implica que, si bien los clústeres obtenidos con Louvain pueden tener mejor cohesión estructural, los del algoritmo greedy garantizan cohesión métrica (distancias cortas).
- **Uso práctico:** Si se busca eficiencia y una segmentación basada en densidad de conexiones (como en redes sociales, redes biológicas, etc.), Louvain es preferible. En cambio, si el objetivo es garantizar tiempos de comunicación bajos dentro de cada grupo (como en redes de sensores o logística), el enfoque basado en bajo diámetro puede ser más adecuado.

Síntesis comparativa:

Criterio	Greedy por Diámetro	Louvain por Modularidad
Control del diámetro	✓	✗
Modularidad alta	✗	✓
Escalabilidad	✗	✓
Determinismo	✓	✗ (depende del orden)
Ejecución en grafos grandes	Limitada	Excelente
Garantía de cohesión métrica	✓	✗

En conclusión, el algoritmo de Louvain representa una excelente heurística para particionar grandes grafos, a costa de no controlar el diámetro. El algoritmo greedy, si bien más costoso, permite establecer cotas sobre la cohesión métrica, actuando como una referencia estructural para evaluar la calidad de soluciones heurísticas como Louvain.

11. Conclusiones

Este trabajo práctico se centró en el Problema de Clustering por Bajo Diámetro, un desafío NP-Completo que busca optimizar la partición de grafos en clústeres. Se demostró formalmente su complejidad intrínseca, confirmando que es NP-Completo mediante una reducción desde el problema de Separación en R Cliques.

Para la resolución, se implementaron un algoritmo de backtracking y un modelo de programación lineal, ambos capaces de encontrar soluciones óptimas en grafos pequeños a medianos. Sus resultados, validados con diversos conjuntos de prueba, demostraron correctitud, aunque con limitaciones de escalabilidad para instancias grandes.

Reconociendo estas limitaciones, se exploró el algoritmo de Louvain, una heurística basada en la modularidad, como una alternativa eficiente para grafos de mayor tamaño, destacando la necesidad de aproximaciones en problemas complejos. La generación de datos de prueba variados fue crucial para evaluar el comportamiento de los distintos algoritmos bajo diversas condiciones.

En síntesis, el trabajo resalta el equilibrio entre la búsqueda de soluciones óptimas en pequeña escala y la aplicación de heurísticas eficientes para abordar la complejidad inherente a los problemas NP-Completos en contextos prácticos.