Optimización de Red de Fibra Óptica (Degree-Constrained Minimum Spanning Tree)

Ernesto Abreu Peraza and Eduardo Brito Labrada

Diseño y Análisis de Algoritmos - Universidad de La Habana

19 de enero de 2026

Resumen

Este informe detalla el análisis e implementa soluciones al conocido problema NP-Hard Degree Constrained Minimum Spanning Tree para la infraestructura de red de la Universidad de La Habana. Se presenta la formalización matemática, la demostración de su complejidad computacional y una comparativa experimental entre diferentes algoritmos.

Índice

1. Introducción

El diseño eficiente de redes de comunicación constituye un problema fundamental en múltiples dominios de la ingeniería y las ciencias de la computación, particularmente en el contexto de infraestructuras físicas como redes de fibra óptica. En estos escenarios, no solo resulta relevante minimizar el costo total de interconexión, sino también respetar restricciones técnicas impuestas por el equipamiento disponible, tales como el número máximo de conexiones que puede soportar cada nodo de la red.

El problema del Árbol de Expansión Mínimo con Restricciones de Grado (*Degree-Constrained Minimum Spanning Tree*, DCMST) surge de manera natural al modelar este tipo de situaciones. A diferencia del problema clásico del Árbol de Expansión Mínimo (MST), el DCMST impone cotas superiores al grado de cada vértice, lo cual incrementa significativamente su complejidad computacional. De hecho, el DCMST es un problema NP-Hard, lo que implica que, salvo que P = NP, no existen algoritmos que lo resuelvan de manera eficiente para grandes instancias.

Motivado por el problema definido en este documento, se realiza una formalización matemática rigurosa del problema, se analizan y se demuestra su complejidad computacional.

Adicionalmente, se diseñan e implementan varios algoritmos y se presenta un análisis experimental que permite comparar el comportamiento de las distintas estrategias implementadas.

2. Formalización del Problema

Partiendo de la necesidad de interconectar los edificios de la universidad minimizando costos y respetando la capacidad de puertos de ETECSA, definimos el modelo matemático.

2.1. Modelo Matemático

Sea $G = (V, E)$ un grafo conexo y no dirigido, donde:

- V es el conjunto de edificios.
- E es el conjunto de posibles conexiones de fibra.
- $w : E \to \mathbb{R}^+$ es una función de costo.
- $k : V \to \mathbb{N}$ es la capacidad de puertos por edificio.

El problema consiste en encontrar un subgrafo $T = (V, E')$ tal que:

$$T^* = \arg\min_{T \in \mathcal{T}} \sum_{e \in E'} w(e) \tag{1}$$

Sujeto a:

1. T es un árbol de expansión de G.
2. $\forall v \in V, \deg_T(v) \leq k(v)$.

Este problema es conocido como *Degree-Constrained Minimum Spanning Tree* (DCMST).

2.2. Variantes del Problema

Analicemos algunas variantes, las cuales surgen a partir de diferentes supuestos sobre la estructura del grafo y las restricciones de grado.

2.2.1. DCMST en grafos completos

Una variante relevante del DCMST es aquella en la que el grafo se asume completo. Formalmente, se considera un grafo no dirigido

$$G = (V, E),$$

donde

$$E = \{\{u, v\} \mid u, v \in V,\ u \neq v\}.$$

Este modelo se ajusta bien al problema original, dado que de no existir una conexión original entre dos vértices (porque sea imposible conectar dos edificios), puedes asignar un costo *infinito* a dicha arista, asegurando que no será seleccionada en la solución óptima. De ser seleccionada representaría que no se encontró solución factible.

Analizando la complejidad temporal, el carácter completo del grafo incrementa significativamente el número de aristas al peor caso,

$$|E| = \frac{|V|(|V| - 1)}{2} = O(|V|^2).$$

2.2.2. DCMST con restricción de grado uniforme

Otra variante ampliamente estudiada es aquella en la que el límite de grado es uniforme para todos los vértices del grafo. En este caso, se fija un valor entero constante $K \geq 2$ tal que

$$\deg_T(v) \leq K \quad \forall v \in V,$$

donde $\deg_T(v)$ denota el grado del vértice v en el árbol generador T.

Esta formulación es adecuada en el caso en que el equipamiento de red sea el mismo para todos los edificios.

Además si tomamos $K = \text{mín}_{v \in V}\, k(v)$, una solución optima de esta instancia es una solución factible, aunque no necesariamente óptima, para la instancia del problema original.

2.2.3. Degree-Constrained Spanning Tree (DCST)

Una variante adicional es el problema Degree-Constrained Spanning Tree. En esta versión, el objetivo no consiste en minimizar el peso total del árbol, sino únicamente en determinar la existencia de un árbol generador que satisfaga las restricciones estructurales.

Este problema puede ser formulado como un problema de decisión o factibilidad. A pesar de la ausencia de una función objetivo de optimización, la determinación de la existencia de una solución válida sigue siendo computacionalmente difícil en el caso general. Esta variante resulta de interés teórico y se emplea frecuentemente como base para el estudio de la complejidad del DCMST y el diseño de algoritmos exactos y aproximados.

En la siguiente sección, se analizará la complejidad computacional del DCMST y sus variantes. En la experimentación nos enfocaremos principalmente analizando DCMST en grafos completos con restricción de grado uniforme por los beneficios planteados anteriormente.

3. Análisis de Complejidad Computacional

3.1. Demostración de NP-Hardness del problema DCMST

Para analizar la complejidad computacional del problema, demostramos que dicho problema es NP-Hard mediante una reducción polinomial desde el problema del Viajante de Comercio (*Traveling Salesman Problem*, TSP), uno de los problemas clásicos NP-Hard.

Teorema 3.1. *El problema Degree-Constrained Minimum Spanning Tree es NP-Hard.*

Demostración. Consideremos una instancia arbitraria del problema del Viajante. Dicha instancia está definida por un grafo completo no dirigido

$$G = (V, E),$$

junto con una función de costos $w : E \to \mathbb{R}^+$. El objetivo del TSP es encontrar un ciclo simple de costo mínimo que visite exactamente una vez cada vértice. Un ciclo simple que cubre todos los vértices es conocido como ciclo hamiltoniano.

A partir de esta instancia, construimos una instancia del problema DCMST de la siguiente manera:

- Obtenemos el mismo grafo $G = (V, E)$ y la misma función de costos w.
- Sea v un vértice cualquiera de G
- Dividimos v en dos vértices v_1 y v_2, tal que $k(v_1) = 1$ y $k(v_2) = 1$.
- Asignamos restricciones de grado $k(u) = 2$ para todo $u \in V \setminus \{v_1, v_2\}$.

Observemos que, bajo esta restricción, cualquier árbol de expansión válido T debe satisfacer

$$\deg_T(v) \leq 2 \quad \forall v \in V.$$

Un árbol de expansión con grado máximo 2 en todos los vértices es necesariamente un camino simple que cubre todos los vértices del grafo. En todo camino simple, los vértices extremos tienen grado 1 y los vértices intermedios tienen grado 2.

Si resolvemos esta instancia, obtenemos un árbol de expansión T^* de costo mínimo que cumple con las restricciones. Por lo que el árbol va a ser de la forma

$$v_1 - u_1 - u_2 - \ldots - u_{|V|-2} - v_2$$

, donde $u_i \in V \setminus \{v_1, v_2\}$.

Lema 3.2. *El árbol de expansión T^* obtenido en la instancia del DCMST corresponde a un ciclo hamiltoniano de costo mínimo en la instancia original del TSP.*

Demostración. Dado el árbol T^*, notese que si unimos nuevamente v_1 y v_2 como un solo vértice v, obtenemos un ciclo hamiltoniano C^* en el grafo original G con costo igual al del árbol T^*. Supongamos que existe un ciclo hamiltoniano C en G con costo menor que el costo de T^*. Al dividir el vértice v en dos vértices v_1 y v_2, el ciclo C se transforma en un camino simple que cubre todos los vértices de la instancia del DCMST, obteniendose un árbol de expansión T' con costo menor que T^*, lo cual es una contradicción ya que T^* es el de costo mínimo.

Por reducción al absurdo, concluimos que C^* es un ciclo hamiltoniano de costo mínimo en la instancia original del TSP. □

Dado que todas las transformaciones realizadas son polinomiales en tiempo, tener una solución de DCMST en tiempo polinomial nos permite resolver cualquier instancia del TSP en tiempo polinomial. Por tanto DCMST es al menos tan difícil como TSP.

Por lo tanto, el problema DCMST es NP-Hard. □

3.2. NP-Completitud del problema Degree-Constrained Spanning Tree

Consideremos ahora la versión de decisión del problema, conocida como Degree-Constrained Spanning Tree (DCST), en la cual se pregunta si existe un árbol de expansión que satisfaga un conjunto dado de restricciones de grado, sin considerar una función de costo.

Teorema 3.3. *El problema Degree-Constrained Spanning Tree (DCST) es NP-Completo.*

Demostración. En primer lugar, observemos que DCST pertenece a la clase NP. En efecto, dada una solución candidata $T = (V, E')$, es posible verificar en tiempo polinomial que:

- T es conexo y acíclico,
- $|E'| = |V| - 1$,
- $\deg_T(v) \leq k(v)$ para todo $v \in V$.

A continuación, demostramos que DCST es NP-Hard mediante una reducción polinomial desde el problema *Hamiltonian Path*, el cual es NP-Completo.

Sea $G = (V, E)$ una instancia arbitraria del problema Hamiltonian Path. Construimos una instancia del problema DCST sobre el mismo grafo G, imponiendo una restricción de grado uniforme $k(v) = 2$ para todo vértice $v \in V$.

G tiene un camino hamiltoniano si y solo si existe un árbol generador T de G tal que $deg_T(v) \leq 2$ para todo $v \in V$.

(⇒) Si G tiene un camino hamiltoniano P que visita todos los vértices exactamente una vez, entonces las aristas de P forman un subgrafo conexo con $|V| - 1$ aristas, por lo que es un árbol generador T. Además, en un camino los extremos tienen grado 1 y los vértices internos grado 2, así que $\deg_T(v) \leq 2$ para todo v.

(⇐) Supongamos ahora que existe un árbol generador T de G tal que $\deg_T(v) \leq 2$ para todo v. Un árbol en el que todos los vértices tienen grado a lo sumo 2 necesariamente es un camino simple que incluye a todos los vértices: si existiera un vértice de grado 3 o más no cumpliría la cota; y si tuviera varios componentes no sería árbol generador. Por lo tanto, T es un camino que visita todos los vértices exactamente una vez, es decir, un camino hamiltoniano en G. □

Como DCST ∈ NP y DCST es NP-Hard, se tiene que DCST es NP-Completo.

4. Diseño de Soluciones Algorítmicas

Se desarrollaron e implementaron cuatro enfoques principales. El código fuente completo de cada implementación se encuentra en el Apéndice A.

4.1. Enumeración con Máscara de Bits para el DCMST

Esta es una solución exacta para el problema DCMST basada en **enumeración exhaustiva mediante máscara de bits**. Esta solución está diseñada para instancias de tamaño pequeño, sirviendo como algoritmo de referencia y verificación de metaheurísticas.

El algoritmo explora todos los subconjuntos posibles de aristas que contienen exactamente $|V| - 1$ aristas y verifica cuáles inducen un árbol generador que cumple con la restricción de grado. Entre todas las soluciones factibles, selecciona la de costo mínimo. La implementación completa se encuentra en el Apéndice **??**.

4.1.1. Descripción del Algoritmo

Sea $G = (V, E)$ un grafo completo, no dirigido y ponderado, con $|V| = n$ vértices y $|E| = m = \dfrac{n(n-1)}{2}$ aristas. El algoritmo procede de la siguiente manera:

1. Se enumeran todas las aristas E y se indexan de 0 a $m - 1$.
2. Se recorre cada máscara de bits $mask \in \{0, 1\}^m$.
3. Solo se consideran las máscaras cuyo número de bits activos es exactamente $n - 1$.
4. Cada máscara define un subgrafo G_{mask} compuesto por las aristas seleccionadas.
5. Se verifica si:
   - El subgrafo es conexo (usando Depth-First Search).
   - El grado de cada vértice es a lo sumo k.
6. Si ambas condiciones se cumplen, el subgrafo es un árbol generador factible y se evalúa su costo.
7. Se devuelve el mínimo costo entre todas las soluciones factibles.

El uso de una máscara de bits permite representar subconjuntos de aristas de forma compacta y eficiente a nivel de implementación.

4.1.2. Análisis de Correctitud

Demostraremos que el algoritmo es correcto, es decir, que devuelve exactamente el costo del árbol generador mínimo con restricción de grado.

Lema 4.1. *El algoritmo examina **todos** los subconjuntos de aristas de tamaño $n - 1$.*

Demostración. Cada máscara de bits m representa un subconjunto único de aristas. Al iterar sobre todas las máscaras con popcount(mask) $= n - 1$, se enumeran exactamente todos los subconjuntos de E con $n - 1$ aristas. □

Lema 4.2. *Un subconjunto de aristas $E' \subseteq E$ con $|E'| = n - 1$ es un árbol generador factible si y solo si:*

1. *El grafo inducido es conexo.*
2. *Para todo vértice v, $deg(v) \leq k$.*

Demostración. Un grafo conexo con $n - 1$ aristas es un árbol. La segunda condición garantiza la restricción de grado. Por lo tanto, ambas condiciones son necesarias y suficientes. □

Lema 4.3. *La función* `check` *devuelve verdadero si y solo si el subgrafo inducido por la máscara es un árbol generador factible.*

Demostración. Inicialmente, verifica que ningún vértice tenga grado mayor que k. Luego, ejecuta un DFS desde el vértice 0 y comprueba que todos los vértices son alcanzables, lo que implica conectividad. Por el Lema 4.2, esto es equivalente a ser un DCST factible. □

Teorema 4.4. *El algoritmo devuelve el costo mínimo entre todos los árboles generadores que cumplen la restricción de grado.*

Demostración. Por el Lema 4.1, el algoritmo considera todas las soluciones candidatas. Por el Lema 4.2, acepta exactamente las soluciones factibles. Finalmente, toma el mínimo costo entre ellas. Por lo tanto, el resultado es óptimo. □

4.1.3. Análisis de Complejidad

El número total de máscaras es 2^m, pero las máscaras consideradas efectivamente son $\binom{m}{n-1}$ y por cada una de ellas se hace lo siguiente:

- Construcción del subgrafo: $O(n)$.
- Verificación de grados: $O(n)$.
- DFS para conectividad: $O(n)$.

Por tanto, el costo por máscara es $O(n)$ y la complejidad total es: $O(2^m + \binom{m}{n-1} \cdot n)$.

4.2. Enumeración Lexicográfica para el DCMST

El algoritmo se basa en la enumeración exhaustiva de subconjuntos de aristas mediante una representación binaria y generación lexicográfica de combinaciones. Su aplicabilidad es sobre todo para instancias de tamaño reducido y es aplicable como algoritmo de referencia para la validación de metaheurísticas. (Ver Apéndice **??**).

4.2.1. Descripción del Algoritmo

Sea $G = (V, E)$ un grafo completo con $|V| = n$ vértices y $|E| = m = \frac{n(n-1)}{2}$ aristas. El algoritmo procede como sigue:

1. Se indexan todas las aristas de E.
2. Se construye una cadena binaria `state` de longitud m con exactamente $n-1$ bits activos.
3. Cada permutación lexicográfica de `state` representa un subconjunto distinto de $n-1$ aristas.
4. Para cada subconjunto:
   - Se construye el subgrafo inducido.
   - Se verifica la restricción de grado.
   - Se comprueba conectividad mediante DFS.
   - Si es factible, se evalúa su costo.
5. Se devuelve el mínimo costo encontrado.

La generación de subconjuntos se realiza mediante la función `next_permutation`, lo que garantiza que cada combinación se visita exactamente una vez.

4.2.2. Análisis de Correctitud

Lema 4.5. *El algoritmo enumera todos los subconjuntos de aristas de tamaño $n-1$.*

Demostración. La cadena binaria inicial contiene exactamente $n-1$ unos y $m-(n-1)$ ceros. El uso de `next_permutation` genera todas las permutaciones distintas de dicha cadena, que corresponden biyectivamente a los subconjuntos de E con $n-1$ elementos. □

Lema 4.6. *Un subconjunto $E' \subseteq E$ con $|E'| = n-1$ es un árbol generador factible si y solo si:*

1. *El subgrafo inducido es conexo.*
2. *Para todo $v \in V$, $\deg(v) \leq k$.*

Demostración. Un grafo conexo con $n-1$ aristas es un árbol. La segunda condición garantiza la restricción de grado. Ambas son necesarias y suficientes. □

Lema 4.7. *La función `check` devuelve verdadero si y solo si el subgrafo inducido por la máscara es un árbol generador factible.*

Demostración. Inicialmente, verifica que ningún vértice tenga grado mayor que k. Luego, ejecuta un DFS desde el vértice 0 y comprueba que todos los vértices son alcanzables, lo que implica conectividad. Por el Lema 4.6, esto es equivalente a ser un DCST factible. □

Teorema 4.8. *El algoritmo devuelve el costo mínimo entre todos los árboles generadores que satisfacen la restricción de grado.*

Demostración. Por el Lema 4.5, todas las soluciones candidatas son evaluadas. Por el Lema 4.6, se aceptan exactamente las factibles. El algoritmo selecciona el mínimo costo entre ellas, lo que implica optimalidad. □

4.2.3. Análisis de Complejidad

El número de combinaciones evaluadas es: $\binom{m}{n-1}$ y para cada combinación:

- Construcción del subgrafo: $O(n)$.
- Verificación de grados: $O(n)$.
- DFS de conectividad: $O(n)$.

Por lo tanto, la complejidad es $O(\binom{m}{n-1} \cdot n)$.

4.3. Enumeración con Código de Gray

A continuación se presenta un algoritmo exacto para el DCMST basados en la enumeración exhaustiva de subconjuntos de aristas utilizando Código de Gray. El enfoque garantiza que subconjuntos consecutivos difieran en exactamente una arista, lo que permite una generación sistemática y eficiente del espacio de soluciones. Se propondrá una solución recursiva con podas.

4.3.1. Marco Teórico: Código de Gray

El código de Gray constituye una secuencia de representaciones binarias tal que dos códigos consecutivos difieren exactamente en un solo bit. Formalmente, el código Gray de un entero n se define como:

$$G(n) = n \oplus (n >> 1)$$

Una técnica para generar todas las combinaciones de tamaño K consiste en generar la secuencia completa de códigos Gray para los enteros desde 0 hasta $2^N - 1$ y filtrar únicamente aquellas máscaras que contienen exactamente K bits activos. Un resultado fundamental es que, en la secuencia filtrada, dos combinaciones adyacentes (consideradas en sentido cíclico) difieren exactamente en dos bits, lo que equivale a eliminar un elemento y añadir otro.

Esta propiedad puede demostrarse por inducción utilizando la construcción recursiva del código Gray:

$$G(N) = 0G(N-1) \cup 1G(N-1)^R$$

4.3.2. Descripción del Algoritmo Gray Recursivo

El algoritmo Gray recursivo implementa directamente la relación:

$$G(N,K) = 0G(N-1,K) \cup 1G(N-1,K-1)^{\mathrm{R}}$$

construyendo únicamente combinaciones válidas de $k = n-1$ aristas. (Ver Apéndice **??**).

Durante la recursión, se mantiene de forma incremental:

- el costo acumulado;
- la estructura de adyacencia;
- las restricciones de grado.

Las ramas que violan la restricción de grado se podan anticipadamente.

4.3.3. Análisis de Correctitud del Algoritmo Gray Recursivo

Lema 4.9. *El algoritmo genera exactamente todas las combinaciones de $n-1$ aristas sin repetición.*

Demostración. Procedemos por inducción fuerte sobre N, la longitud de la cadena binaria (número total de aristas disponibles).

Caso base: Para $N = 0$, $G(0,0) = \{\epsilon\}$, que representa la única combinación de 0 aristas. Para $K > 0$, $G(0,K) = \emptyset$, correctamente generando ninguna combinación.

Hipótesis inductiva: Supongamos que para todo $n' < N$, $G(n',K)$ genera exhaustiva y sin repeticiones todas las combinaciones binarias de longitud n' con exactamente K bits activos.

Paso inductivo: Para $G(N,K)$, consideremos cualquier combinación válida s de longitud N con K bits 1. Examinamos su primer bit:

- Si $s_1 = 0$, entonces el sufijo $s_{2..N}$ es una combinación válida de longitud $N-1$ con K bits 1. Por hipótesis, aparece en $G(N'1,K)$, luego $0s_{2..N}$ aparece en $0G(N-1,K)$.
- Si $s_1 = 1$, entonces $s_{2..N}$ tiene $K-1$ bits 1. Por hipótesis, aparece en $G(N-1,K-1)$, luego $1s_{2..N}$ aparece en $1G(N-1,K-1)^R$.

La unión disjunta $\cup$ de ambos conjuntos cubre todas las posibilidades sin superposición, pues difieren en su primer bit.

Por tanto, $G(N,K)$ genera exactamente todas las combinaciones buscadas, sin duplicados. □

Lema 4.10. *Toda solución considerada por el algoritmo cumple la restricción de grado.*

Demostración. Antes de continuar la recursión, el algoritmo verifica que el grado de los vértices involucrados no exceda K. Las ramas inválidas son descartadas. □

Teorema 4.11. *El algoritmo Gray recursivo produce una solución óptima del DCMST.*

Demostración. Por el Lema 4.9 y por el Lema 4.10 el algoritmo explora exhaustivamente el espacio de soluciones factibles y conserva el mínimo costo encontrado, garantizando optimalidad. □

4.4. Enumeración Exhaustiva con Gosper's hack

4.4.1. Descripción del algoritmo

Sea E el conjunto de aristas del grafo, construido tomando todos los pares $i < j$ con peso $\texttt{mat}[i][j]$. El algoritmo enumera **todos** los subconjuntos de aristas $E' \subseteq E$ con cardinalidad exactamente $|E'| = V-1$. Para cada subconjunto:

1. Construye el subgrafo inducido $G' = (V,E')$ mediante listas de adyacencia.
2. Calcula el costo total $\sum_{e \in E'} w(e)$.

3. Verifica factibilidad mediante `check(V,K,adj)`, que comprueba:
   - que G' sea un árbol generador (conexo y acíclico), y
   - que $\deg_{G'}(v) \leq K$ para todo $v \in V$.
4. Si es factible y su costo es menor que el mejor registrado, actualiza la solución óptima.

La enumeración se realiza usando una **máscara de bits** `mask` de 64 bits que indica qué aristas de E están seleccionadas. Para iterar eficientemente por todas las máscaras con exactamente $V-1$ bits en 1, se utiliza la técnica conocida como *Gosper's hack*, que genera en tiempo $O(1)$ la siguiente combinación de tamaño fijo.

$$\begin{aligned} c &= mask \ \& \ (-mask) \\ r &= mask + c \\ mask' &= r \mid \left(\frac{(r \oplus mask) \gg 2}{c} \right). \end{aligned}$$

Intuitivamente, c aísla el bit menos significativo encendido, r produce el "acarreo" que mueve hacia la izquierda el bloque adecuado de bits, y el último término recoloca los bits restantes en las posiciones menos significativas para mantener el conteo de unos constante.

4.4.2. Correctitud

Lema 4.12 (Exhaustividad)**.** *El algoritmo considera todos los subconjuntos $E' \subseteq E$ tales que $|E'| = V-1$.*

Demostración. La variable `mask` codifica subconjuntos de E. Se inicializa con $V-1$ bits activados y se actualiza usando *Gosper's hack*, que genera sin repetición todas las máscaras de longitud $|E|$ con exactamente $V-1$ bits en 1. Por tanto, se recorren todas las combinaciones posibles de $V-1$ aristas. □

Lema 4.13 (Filtrado de factibilidad)**.** *Una combinación E' es aceptada si y solo si induce un árbol generador que respeta la cota de grado* $\deg(v) \leq K$.

Demostración. La función `check(V,K,adj)` verifica precisamente (i) que el subgrafo sea un árbol generador y (ii) que se cumpla la cota de grado. Por consiguiente, el algoritmo acepta exactamente las combinaciones factibles. □

Teorema 4.14 (Optimalidad)**.** *Si existe al menos una solución factible, el algoritmo devuelve una solución factible de costo mínimo.*

Demostración. Por el Lema de Exhaustividad, toda solución candidata con $V-1$ aristas es evaluada. Por el Lema de Factibilidad, únicamente las soluciones factibles pueden actualizar el óptimo. El algoritmo mantiene el mínimo costo entre todas las soluciones factibles examinadas; luego, el resultado final es óptimo. □

4.5. Complejidad

Sea $n = V$ y $m = |E| = \frac{n(n-1)}{2}$. El número de subconjuntos evaluados es:

$$\binom{m}{n-1}.$$

Para cada subconjunto, el programa recorre todas las m aristas para construir E' y calcular el costo, lo que cuesta $O(m)$, más el costo de `check`, típicamente $O(n)$ al operar sobre un subgrafo con $n-1$ aristas. Por tanto, una cota superior del tiempo total es:

$$O\left(\binom{m}{n-1} \cdot (m+n) \right),$$

lo cual es combinatorio y solo viable para instancias pequeñas.

El uso de memoria es $O(n^2)$ por la matriz de costos, $O(m)$ por el vector de aristas y $O(n)$ por las adyacencias del subgrafo actual.

5. Diseño de metaheurísticas para resolver el DCMST

5.1. Método Primal Aleatorizado (RPM)

El problema del Árbol de Expansión Mínima con Restricción de Grado (DCMST) busca encontrar un subgrafo T de un grafo ponderado $G = (V, E)$ tal que T sea un árbol que cubra todos los vértices en V, minimice la suma de los pesos de las aristas y satisfaga la restricción de que ningún vértice tenga un grado mayor a d.

Matemáticamente, buscamos minimizar:

$$Z(x) = \sum_{e \in E} w_e x_e \tag{2}$$

Sujeto a:

$$\sum_{e \in \delta(v)} x_e \leq d, \quad \forall v \in V \tag{3}$$

donde $x_e \in \{0, 1\}$ indica si la arista e está en el árbol, y $\delta(v)$ es el conjunto de aristas incidentes en v. A diferencia del MST clásico, el DCMST es un problema NP-hard para $2 \leq d < |V| - 1$.

La solución implementada utiliza el *Randomised Primal Method* (RPM). Este enfoque híbrido combina la eficiencia constructiva del algoritmo de Prim con la capacidad de exploración de un Algoritmo Genético (AG).

5.1.1. Codificación del Cromosoma

En lugar de codificar aristas directamente (lo cual generaría muchas soluciones infactibles), el RPM utiliza una codificación indirecta. Un cromosoma es una matriz C de dimensiones $n \times (d-1)$, donde n es el número de nodos.

El alelo $C[i][k]$ representa un índice de preferencia. Específicamente, indica que cuando el algoritmo de construcción está considerando el nodo i y este tiene actualmente un grado k, debe seleccionar la $C[i][k]$-ésima mejor arista disponible incidente a i.

5.1.2. Decodificación (Algoritmo de Prim Modificado)

El decodificador construye el árbol iterativamente:

1. Se inicia con un vértice arbitrario en el árbol S.
2. En cada paso, se identifican todas las aristas válidas que conectan un nodo $u \in S$ con un nodo $v \notin S$, siempre que $grado(u) < d$.
3. Para cada nodo $u \in S$, en lugar de elegir obligatoriamente la arista de menor peso (como en Prim clásico), se consulta el cromosoma. Se selecciona la arista en la posición $C[u][grado\_actual(u)]$ de su lista de adyacencia ordenada.
4. De este subconjunto de aristas candidatas seleccionadas por el cromosoma, se elige la de menor peso global para añadirla al árbol.

5.1.3. Operadores Genéticos e Inicialización

- **Inicialización Sesgada:** Para asegurar que la búsqueda comience en regiones prometedoras, los alelos no se generan uniformemente. Se utiliza una distribución geométrica (o exponencial negativa discreta) para favorecer valores bajos $(0, 1)$. Esto significa que el algoritmo tiende a comportarse como el algoritmo de Prim (codicioso) con pequeñas perturbaciones estocásticas.
- **Cruce:** Se implementa un cruce uniforme (*Uniform Crossover*), donde cada gen del hijo se toma de uno de los padres con probabilidad 0.5.
- **Mutación:** Se aplica una mutación puntual con baja probabilidad, reasignando el valor del gen utilizando la misma distribución geométrica sesgada de la inicialización.

5.1.4. Manejo de Soluciones Inválidas

Dado que las restricciones de grado pueden, en ciertas topologías, llevar a callejones sin salida donde no es posible conectar más nodos sin violar restricciones, el decodificador incluye un mecanismo de seguridad. Si el árbol no logra conectar los n nodos, la solución se marca como `valid = false` y se le asigna un costo infinito, siendo descartada por el proceso de selección del AG.

5.2. Colonia de Hormigas (ACS)

La implementación sigue el paradigma del *Ant Colony System* (ACS). El algoritmo construye soluciones iterativamente mediante agentes (hormigas) que recorren el grafo tomando decisiones probabilísticas basadas en dos parámetros: la feromona (τ) y la heurística (η).

5.2.1. Modelo de Decisión

Para una hormiga situada en el proceso de construcción, la elección de la siguiente arista (u, v) para añadir al árbol se basa en la "atractividad" de la arista, definida como:

$$Attr_{uv} = [\tau_{uv}]^{\alpha} \cdot [\eta_{uv}]^{\beta} \tag{4}$$

Donde:

- τ_{uv}: Nivel de feromona en la arista (u, v), representando la experiencia acumulada de la colonia.
- η_{uv}: Información heurística, definida inversamente proporcional al peso de la arista (w_{uv}):

$$\eta_{uv} = \frac{1}{w_{uv} + \epsilon} \tag{5}$$

- α, β: Parámetros que controlan la influencia relativa de la feromona y la heurística.

5.2.2. Regla de Transición Pseudo-Aleatoria

El algoritmo implementa una regla de decisión híbrida controlada por el parámetro q_0 (probabilidad de explotación):

Si se genera un número aleatorio $q \in [0, 1]$ tal que $q \leq q_0$, la hormiga elige determinísticamente la mejor arista:

$$(u, v)^* = \arg \max_{(u,v) \in \mathcal{C}} \{Attr_{uv}\} \tag{6}$$

En caso contrario ($q > q_0$), la hormiga realiza una selección probabilística (exploración) tipo ruleta, donde la probabilidad P_{uv} de elegir la arista (u, v) es:

$$P_{uv} = \frac{Attr_{uv}}{\sum_{(i,j) \in \mathcal{C}} Attr_{ij}} \tag{7}$$

Donde $\mathcal{C}$ es el conjunto de aristas candidatas factibles (aquellas que no cierran ciclos y cumplen la restricción de grado K).

5.2.3. Actualización de Feromonas

La actualización de feromonas sigue un enfoque elitista para acelerar la convergencia.

- **Evaporación:** Al final de cada iteración global, la feromona en todas las aristas se evapora para evitar óptimos locales:

$$\tau_{uv} \leftarrow \tau_{uv} \cdot (1 - \rho) \tag{8}$$

- **Depósito de Feromona:** Se realiza un refuerzo basado en las mejores soluciones encontradas:

  1. **Mejor Local:** La mejor hormiga de la iteración actual deposita feromona.

2. **Mejor Global:** La mejor solución encontrada desde el inicio del algoritmo deposita un refuerzo adicional.

El incremento de feromona $\Delta\tau$ es inversamente proporcional al costo total del árbol (C_{best}):

$$\tau_{uv} \leftarrow \tau_{uv} + \frac{\text{Constante}}{C_{best}} \tag{9}$$

6. Análisis Experimental

6.1. Implementación

Con el objetivo de evaluar los algoritmos propuestos para la resolución del problema DCMST, se implementaron todas las soluciones descritas en la sección anterior utilizando el lenguaje de programación `C++`. La elección de este lenguaje se debe a su eficiencia, amplia disponibilidad de estructuras de datos de bajo nivel y la experiencia de los autores en el mismo, lo cual hace mas sencilla la implementación de algoritmos.

6.2. Intancias de Prueba

La evaluación se hará por grupos. Cada grupo contiene un conjunto de instancias de prueba con un valor de 100 puntos cada uno. La puntuación para un grupo será la suma de los puntos obtenidos en cada instancia de ese grupo.

Los puntos serán otorgados dependiendo de qué tan cerca esté la solución del mínimo costo sin tener en cuenta la restricción de k o de una solución exacta encontrada previamente para esa instancia, sea mst_cost este mínimo costo y sea sol_cost el costo encontrado por el algoritmo a evaluar; la puntuación para esa instancia será:

$$100 \times \frac{\text{mst\_cost}}{\text{sol\_cost}}$$

A continuación se muestra la descripción detallada sobre cada grupo:

Grupos	**Restricciones adicionales**		**Puntos**	**Tests**
	n	$a_{i,j}$		
`maxN6`	$2 \leq n \leq 6$	—	2500	25
`maxN9`	$7 \leq n \leq 9$	—	3000	30
`N10`	$n = 10$	—	1000	10
`N11`	$n = 11$	—	1000	10
`N12`	$n = 12$	—	1500	15
`N13`	$n = 13$	—	1500	15
`N14`	$n = 14$	—	1500	15
`N15`	$n = 15$	—	1500	15
`random1`	$16 \leq n \leq 30$	—	3000	30
`random2`	$31 \leq n \leq 50$	—	3000	30
`random3`	$51 \leq n \leq 70$	—	5000	50
`random4`	$71 \leq n \leq 100$	—	5000	50

Las instancias de prueba utilizadas corresponden a grafos completos de distintos tamaños representados en matriz de adyacencia ponderada. Cada instancia tiene el siguiente formato:

- La primera línea contiene dos enteros n y k ($1 \leq k < n \leq 100$) — la cantidad de edificios y el número máximo de conexiones que soporta cada edificio, respectivamente.
- Las siguientes n líneas contienen cada una n enteros. El entero j de la fila i, denotado por $a_{i,j}$ ($0 \leq a_{i,j} \leq 100$) representa el costo de instalar un cable de fibra óptica entre el edificio i y el edificio j.
- Se garantiza que $a_{i,i} = 0$ para todo $1 \leq i \leq n$ y que $a_{i,j} = a_{j,i}$ para todo $1 \leq i, j \leq n$.

La salida de cada algoritmo debe tener el siguiente formato:

- La primera línea debe contener un entero c — el mínimo costo posible.
- Le deben seguir $n - 1$ líneas. La línea i de ellas debe contener tres enteros u_i, v_i y w_i ($1 \leq u_i, v_i \leq n$, $u_i \neq v_i$, $1 \leq w_i \leq 100$) — existe una ruta entre los edificios u_i y v_i con costo w_i. La instalación formada por estas rutas debe cumplir con las condiciones del problema.

6.3. Experimentación

Para cada instancia se evaluó el costo mínimo encontrado, así como el tiempo de ejecución de cada algoritmo, permitiendo comparar su desempeño relativo.

El propósito principal de este análisis experimental no es demostrar escalabilidad, sino:

- Validar la correctitud de las implementaciones.
- Comparar el impacto de distintas estrategias sobre el tiempo de ejecución.

La experimentación se realizó en las plataformas Codeforces y Polygon. El tiempo límite máximo de ejecución para cada algoritmo fue de 15000ms y una memoria límite máxima de 1024mb.

6.4. Resultados

A continuación se presentan los resultados obtenidos para los tres algoritmos deterministas implementados, esta tabla muestra la cantidad de puntos obtenidos por cada algoritmo y el tiempo límite máximo en cada grupo.

Grupos	**bitmask**		**comb/gosper**		**gray_recursive**	
	Puntos	**Tiempo máx**	**Puntos**	**Tiempo máx**	**Puntos**	**Tiempo máx**
`maxN6`	2500	31	2500	46	2500	46
`maxN9`	2000	15000	3000	14890	3000	7406
`N10`	0	15000	0	15000	148	15000
`N11`	—	—	—	—	0	15000
`N12`	—	—	—	—	—	—
`N13`	—	—	—	—	—	—
`N14`	—	—	—	—	—	—
`N15`	—	—	—	—	—	—
`random1`	—	—	—	—	—	—
`random2`	—	—	—	—	—	—
`random3`	—	—	—	—	—	—
`random4`	—	—	—	—	—	—
`total`	4500	15000	5500	15000	5648	15000

De acuerdo con la tabla de comparación de algoritmos (`bitmask`, `comb/gosper` y `gray_recursive`) se observan los siguientes resultados:

- **Cobertura de grupos:** `bitmask` y `comb/gosper` funcionan correctamente únicamente en los grupos más pequeños (`maxN6` y `maxN9`). `gray_recursive` es el único algoritmo que obtiene resultados en los grupos medianos (`N10` y `N11`), aunque con tiempos máximos cercanos al límite permitido. Ninguno de los algoritmos logró resolver los grupos más grandes (`N12`–`N15` y `random1`–`random4`), mostrando limitaciones de escalabilidad.
- **Puntos obtenidos:** En `maxN6`, los tres algoritmos alcanzan la puntuación máxima (2500 puntos), indicando desempeño completo en instancias pequeñas. Para `maxN9`, `bitmask` alcanza solo 2000 puntos, mientras que `comb/gosper` y `gray_recursive` logran 3000 puntos, mostrando que `bitmask` pierde eficiencia en instancias medianas. En `N10`, únicamente `gray_recursive` obtuvo 148 puntos, mientras que los otros dos algoritmos no lograron puntos.
- **Tiempo máximo:** Todos los algoritmos alcanzaron o se aproximaron al tiempo límite (15000 ms) en las instancias más grandes y medianas, lo que indica que estas implementaciones no escalan bien y que los tiempos son críticos para instancias mayores a $n = 9$.
- **Resumen total:** Sumando los puntos obtenidos por cada algoritmo, `gray_recursive` alcanza la puntuación más alta total (5648 puntos), seguido por `comb/gosper` (5500) y `bitmask` (4500). Esto refleja que `gray_recursive` es más robusto frente a instancias medianas, aunque sigue limitado en instancias grandes.

A continuación se presentan los resultados obtenidos para los algoritmos heurísticos implementados, esta tabla muestra la cantidad de puntos obtenidos por cada algoritmo y el tiempo límite máximo en cada grupo.

Grupos	evolutionary		ant_colony200		ant_colony100	
	Puntos	Tiempo máx	Puntos	Tiempo máx	Puntos	Tiempo máx
maxN6	2500	140	2500	203	2500	93
maxN9	2990	265	3000	281	2995	171
N10	921	359	945	375	945	234
N11	895	375	970	531	970	265
N12	1275	421	1429	687	1424	390
N13	1321	468	1417	906	1416	484
N14	1302	531	1450	1187	1450	609
N15	1197	609	1426	1593	1426	765
random1	2314	2453	2945	12625	2941	7187
random2	1691	7828	233	15000	1230	15000
random3	2342	15000	0	15000	0	15000
random4	20	15000	0	15000	0	15000
total	18768	15000	16315	15000	17297	15000

Grupos	ant_colony50		ant_colony25		ant_colony12	
	Puntos	Tiempo máx	Puntos	Tiempo máx	Puntos	Tiempo máx
maxN6	2500	93	2500	46	2500	31
maxN9	2995	109	2995	62	2995	62
N10	945	140	945	93	945	62
N11	970	156	970	125	967	62
N12	1424	203	1424	125	1422	78
N13	1417	265	1417	156	1409	93
N14	1450	312	1447	187	1446	109
N15	1426	406	1426	203	1426	125
random1	2942	3578	2940	1765	2937	937
random2	2225	15000	2925	9250	2924	4546
random3	0	15000	2262	15000	4955	12984
random4	0	15000	0	15000	356	15000
total	18294	15000	21251	15000	24282	15000

Se presentan los resultados de comparación de los algoritmos `evolutionary` y variantes de `ant_colony` con diferentes parámetros de población.

- **Cobertura de grupos:** Todos los algoritmos obtienen resultados completos en los grupos más pequeños (`maxN6` y `maxN9`). A medida que el tamaño de las instancias aumenta, los algoritmos comienzan a diferenciarse: `evolutionary` mantiene puntuaciones consistentes hasta `N15` y `random1`, pero pierde eficacia en `random2-random4`. Las versiones de `ant_colony` muestran un comportamiento dependiente del tamaño de población: las colonias mayores (200, 100) alcanzan mejores resultados en instancias medianas, mientras que las colonias más pequeñas (50, 25, 12) obtienen puntuaciones más bajas en los grupos grandes.

- **Puntos obtenidos:**

  - En los grupos pequeños y medianos (`maxN6`–`N15`), todos los algoritmos alcanzan puntuaciones altas y consistentes, aunque `ant_colony200` y `ant_colony100` superan ligeramente a `evolutionary` en algunos casos (`N12-N15`).
  - Para instancias grandes y aleatorias (`random1-random4`), la puntuación total aumenta conforme se incrementa la población de la colonia de hormigas: por ejemplo, `ant_colony12` obtiene 24282 puntos totales frente a 18294 de `ant_colony50`.

- En general, `ant_colony12` y `ant_colony25` muestran que con una configuración adecuada se pueden superar los resultados de `evolutionary` en instancias grandes.

- **Tiempo máximo:**
  - Los tiempos de `evolutionary` son moderados en instancias pequeñas y medianas, pero alcanzan 15000 ms (tiempo límite) en las instancias más grandes (`random3-random4`).
  - Las versiones de `ant_colony` muestran un comportamiento dependiente de la población: colonias grandes tardan más (hasta 15000 ms en algunos casos), mientras que colonias pequeñas reducen significativamente el tiempo, aunque algunas puntuaciones disminuyen.
  - Existe un trade-off claro entre tamaño de población y tiempo de ejecución: mayor población → mejores resultados pero mayor tiempo; menor población → resultados más rápidos pero potencialmente menores.

- **Resumen total:**
  - En puntuación total acumulada, `ant_colony12` es el algoritmo más efectivo (24282 puntos), seguido por `ant_colony25` (21251) y `evolutionary` (18768).
  - Esto indica que, aunque `evolutionary` es estable y consistente, las variantes de colonia de hormigas bien parametrizadas superan el rendimiento en instancias grandes.

7. Conclusiones

En este trabajo se detalló el análisis y solución del problema DCMST. Se demostró su complejidad NP-Hard, se presentaron cuatro variantes algorítmicas exactas y dos metaheurísticas, que fueron evaluadas y comparadas sobre las mismas intancias de prueba. Los algoritmos exactos mostraron limitaciones de escalabilidad, siendo viables solo para instancias pequeñas y medianas. Entre ellos, el enfoque basado en código de Gray recursivo con poda fue el más robusto, alcanzando soluciones óptimas en instancias de hasta 11 nodos. Las metaheurísticas demostraron ser más efectivas para instancias grandes. El método primal aleatorizado (RPM) y las colonias de hormigas (ACS) lograron soluciones cercanas al óptimo en tiempos razonables. En particular, las variantes de ACS con poblaciones ajustadas mostraron un balance favorable entre calidad de solución y tiempo de ejecución.

Referencias

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed., Chapter 34). MIT Press.

[2] J. Knowles and D. Corne. 2000. A new evolutionary approach to the degree-constrained minimum spanning tree problem. Trans. Evol. Comp 4, 2 (July 2000), 125-134.

[3] Wanru Gao, Mojgan Pourhassan, Vahid Roostapour, and Frank Neumann. 2019. Runtime Analysis of Evolutionary Multi-objective Algorithms Optimising the Degree and Diameter of Spanning Trees. In Evolutionary Multi-Criterion Optimization: 10th International Conference, EMO 2019, East Lansing, MI, USA, March 10-13, 2019, Proceedings. Springer-Verlag, Berlin, Heidelberg, 504-515.