Universidad Nacional Autónoma de México
Facultad de Ciencias
Complejidad Computacional

K-Arbol Generador de peso mínimo con heuristica inspirada en PSO

Estudiantes :
Ortiz Montiel Diego Iain
319072369

Profesor :
Canek Peláez Valdés

Ayudantes :
Leslie Ramírez Gallegos

Índice

Resumen

1. Introducción

El problema del **árbol generador mínimo k-acotado** (*k-Minimum Spanning Tree Problem*, k-MST) consiste en encontrar un árbol que conecte exactamente k vértices de una gráfica ponderada, de manera que el peso total de las aristas incluidas sea mínimo. [1], [2] Este problema generaliza el clásico *Minimum Spanning Tree* (MST), que busca el árbol de peso mínimo que conecta todos los vértices de la gráfica. [3]

El k-MST tiene aplicaciones directas en contextos donde se busca establecer infraestructuras parciales de costo mínimo, tales como el diseño de redes de comunicación con presupuesto limitado, la selección de ubicaciones óptimas para estaciones de servicio o centros de distribución, y la planificación de subredes en sistemas de transporte o energía. Un ejemplo claro de lo anterior sería que una empresa de telecomunicaciones puede buscar conectar a sus k clientes más rentables mediante una red de fibra óptica, minimizando el costo total de cableado. En todos estos casos, solo una fracción de los nodos puede ser conectada debido a restricciones de recursos, lo que exige encontrar subconjuntos de vértices que ofrezcan una conectividad eficiente. [1]

—

2. Formulación del problema

2.1. Definición formal

Sea $G = (V, E, w)$ una gráfica no dirigida y ponderada, donde:

- V es el conjunto de vértices, con $|V| = n$;
- $E \subseteq V \times V$ es el conjunto de aristas; y
- $w : E \to R^+$ es una función de pesos asociada a cada arista.

Dado un entero k tal que $2 \leq k \leq n$, el **problema del árbol generador mínimo k-acotado** (*k-Minimum Spanning Tree Problem*, k-MST) consiste en encontrar una subgrafica $T = (V_T, E_T) \subseteq G$ que satisfaga:

1. $V_T \subseteq V$ y $|V_T| = k$;
2. T es un árbol (conexo y acíclico);
3. $E_T \subseteq E$;
4. el **costo total** de T,
$$W(T) = \sum_{(u,v)\in E_T} w(u, v),$$
es mínimo entre todos los subárboles de G que contienen exactamente k vértices.

Formalmente,

$$T^* = \arg\min_{T \subseteq G} \left\{ \sum_{(u,v)\in E_T} w(u,v) \ : \ T \text{ es árbol y } |V_T| = k \right\}.$$

—

2.2. Complejidad del k-MST

Podemos distinguir dos versiones del problema: **la versión de decisión** y **la versión de optimización**.

Versión de decisión. Dada una grafica ponderado $G = (V, E, w)$, un entero k y un umbral B, preguntamos si existe un subárbol $T = (V_T, E_T)$ tal que $|V_T| = k$, T sea árbol, y $\sum_{(u,v)\in E_T} w(u,v) \leq B$. Esta versión **pertenece a NP**, ya que dada una solución candidata, puede verificarse en tiempo polinomial que: (i) $|V_T| = k$, (ii) T es conexo y acíclico (por ejemplo, mediante un recorrido BFS o DFS), y (iii) el costo total no excede B. Además, es **NP-completa**, por reducciones estándar desde problemas NP-duros relacionados, como el *Minimum Connected Subgraph* o el *Steiner Tree Problem*.

Versión de optimización. La versión de optimización, definida previamente, busca el árbol de costo mínimo entre todos los subárboles de tamaño k. Aunque una solución candidata puede verificarse en tiempo polinomial, **no puede comprobarse su optimalidad sin explorar potencialmente todas las combinaciones posibles de subárboles de tamaño** k. Por tanto, esta versión es **NP-hard**, pero **no pertenece a NP**, ya que verificar que una solución es óptima no es posible en tiempo polinomial (a menos que $P = NP$). [3]

—

2.3. Tamaño del espacio de búsqueda y complejidad combinatoria

El carácter intrínsecamente difícil del k-MST puede comprenderse mejor al analizar el tamaño del espacio de búsqueda asociado. Dada una gráfica con n vértices, existen exactamente

$$\binom{n}{k}$$

posibles subconjuntos de vértices de tamaño k. Para cada uno de esos subconjuntos $V_T \subseteq V$, deben considerarse todos los posibles árboles que pueden formarse con esos k vértices, cuyo número crece de manera superexponencial según la fórmula de Cayley:

$$k^{k-2}.$$

Por tanto, el número total de árboles posibles podrían evaluarse en una búsqueda exhaustiva es del orden de:

$$O\left(\binom{n}{k} \cdot k^{k-2}\right),$$

lo cual hace que incluso instancias moderadas sean inabordables mediante fuerza bruta. Por ejemplo, para $n = 200$ y $k = 10$, el número de subconjuntos posibles supera los $2{,}24 \times 10^{16}$, sin considerar aún las combinaciones internas de aristas que forman árboles válidos.

En consecuencia, el **espacio de búsqueda crece combinatoriamente tanto en función de** n (el tamaño total de la gráfica) **como de** k (el número de vértices que deben conectarse). Lo anterior justifica la necesidad de emplear heurísticas capaces de explorar de manera inteligente este espacio, evitando, de esta forma, la enumeración exhaustiva de soluciones y encontrando configuraciones cercanas al óptimo global en tiempo razonable.

3. Heurística de optimización por enjambre de partículas (PSO)

3.1. Metáfora y principio de funcionamiento

El algoritmo *Particle Swarm Optimization* (PSO) es una metaheurística poblacional inspirada en el comportamiento colectivo de los enjambres en la naturaleza, como las bandadas de aves o los bancos de peces. Cada partícula representa una solución potencial del problema y se desplaza en el espacio de búsqueda influenciada por su experiencia personal y por la mejor experiencia conocida por el grupo.

Así pues, de forma análoga a un enjambre que explora su entorno en busca de alimento,las partículas se comunican indirectamente a través del conocimiento del mejor individuo encontrado hasta el momento. Esto permite un equilibrio entre **exploración** (búsqueda de nuevas regiones del espacio) y **explotación** (mejoramiento de soluciones prometedoras).

El algoritmo de Optimización por Enjambre de Partículas (PSO) fue propuesto originalmente por Kennedy y Eberhart en 1995 [4], inspirado en la dinámica colectiva de aves y peces. Posteriormente, Clerc y Kennedy introdujeron un modelo matemático que garantiza estabilidad y convergencia [5]. Las versiones discretas del PSO fueron estudiadas extensamente en la literatura para su aplicación a problemas combinatorios y de graficas [6], [7], incluyendo variantes exitosas para el k-MST [1].

3.2. PSO clásico continuo

En su forma original, el PSO opera sobre espacios continuos: cada partícula tiene una posición $\mathbf{x}_i \in R^n$ y una velocidad $\mathbf{v}_i \in R^n$, que se actualizan en cada iteración como:

$$\mathbf{v}_i(t+1) = \omega \mathbf{v}_i(t) + \alpha_p r_p\big(\mathbf{p}_i - \mathbf{x}_i(t)\big) + \alpha_g r_g\big(\mathbf{g} - \mathbf{x}_i(t)\big),$$

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1),$$

donde:

- $\mathbf{p}_i$ es la mejor posición personal de la partícula i,

- **g** es la mejor posición global del enjambre,
- α_p, α_g son los coeficientes de atracción personal y global,
- $r_p, r_g \sim U(0,1)$ son factores de aleatoriedad, y
- ω es un coeficiente de inercia que regula la estabilidad del movimiento.

3.3. Adaptación al caso discreto del k-MST

En el problema del k-MST, el espacio de búsqueda no es continuo, sino discreto: cada solución es un subconjunto de vértices

$$S_i \subseteq V, \quad |S_i| = k.$$

Por tanto, no tiene sentido hablar de velocidades ni desplazamientos vectoriales. Así mismo, para adaptar el PSO, se definió una **transición discreta por conjuntos**, siguiendo el principio introducido por Kennedy y Eberhart en su versión binaria del PSO [8], donde las partículas evolucionan sobre espacios discretos mediante operadores de reemplazo basada en tres influencias:

- $A = gbest \setminus current$: vértices en la mejor solución global que no están en la actual;
- $B = pbest \setminus current$: vértices en la mejor solución personal que no están en la actual;
- $C = V \setminus (current \cup gbest \cup pbest)$: vértices no explorados en ninguna de las soluciones anteriores.

En cada iteración, cada partícula elige aleatoriamente un vértice nuevo a partir de uno de esos conjuntos (según probabilidades $\alpha_g, \alpha_p, 1-(\alpha_g+\alpha_p)$) y reemplaza uno existente en su conjunto actual. La nueva solución se evalúa calculando su árbol generador mínimo sobre la subgráfica inducido por los k vértices seleccionados.

3.4. Adaptación de la función de costo

Por motivos de optimización en la búsqueda, nos conviene trabajar con una gráfica completa, de modo que todo par de vértices esté conectado mediante una arista. Para lograrlo, redefinimos la función de costo $f(u,v)$ de la siguiente manera:

$$f(u,v) = \begin{cases} w(u,v), & \text{si } (u,v) \in E, \\ d(u,v) \times \operatorname{diam}(\mathcal{G}) \times k, & \text{si } (u,v) \notin E. \end{cases}$$

De esta forma, cada arista agregada artificialmente (es decir, aquellas que no pertenecen a la gráfica original) resulta siempre más pesada que cualquier arista existente, garantizando así la factibilidad de las soluciones que solo utilizan aristas reales.

3.5. Justificación del normalizador como cota superior

El normalizador se define como la suma de las $k-1$ aristas más pesadas de la gráfica original:

$$N = \sum_{i=1}^{k-1} w_{(i)},$$

donde $w_{(1)} \geq w_{(2)} \geq \cdots \geq w_{(m)}$ son los pesos de las aristas ordenadas en forma descendente.

Un árbol con k vértices contiene exactamente $k-1$ aristas, por lo tanto, el costo total $W(T)$ de cualquier árbol factible se obtiene sumando $k-1$ pesos de aristas que pertenecen al conjunto original E. Dado que todas esas aristas son menores o iguales que las $k-1$ más pesadas del conjunto completo, se cumple que:

$$W(T) \leq N.$$

Por construcción, esto implica que el normalizador N es una **cota superior estricta** para el peso de cualquier árbol que utilice únicamente aristas reales de la gráfica. Así, para toda solución factible se cumple:

$$\frac{W(T)}{N} \leq 1.$$

Si en cambio el árbol incluye al menos una arista artificial —cuyo costo fue definido como proporcional al diámetro de la gráfica y, por tanto, estrictamente mayor que cualquier peso real— entonces su peso total $W(T)$ excederá inevitablemente a N, de modo que:

$$\frac{W(T)}{N} > 1.$$

Por consiguiente, el cociente $W(T)/N$ actúa simultáneamente como una **medida de factibilidad** y como un indicador de calidad relativa:

- Si $W(T)/N < 1$, la solución es factiblex.
- Si $W(T)/N > 1$, la solución contiene al menos una arista no existente en la gráfica original.

Esta propiedad permite evaluar la factibilidad de cada árbol generado sin necesidad de realizar comprobaciones adicionales de conectividad o validez estructural.

4. Heurística PSO Discreta para el Problema k-MST

4.1. Introducción general

El algoritmo de *Particle Swarm Optimization* (PSO) se inspira en el comportamiento colectivo de enjambres naturales, como bandadas de aves o cardúmenes de peces, donde los individuos cooperan indirectamente a través de la observación del éxito de los demás.

Cada partícula representa una posible solución del problema y se mueve dentro del espacio de búsqueda guiada por dos componentes principales: su propia experiencia (p_{best}) y la del grupo (g_{best}).

En el caso del problema k-MST, el PSO debe adaptarse a un espacio de búsqueda discreto, donde cada solución es un subconjunto de k vértices de la gráfica. A continuación se describen los principales componentes y adaptaciones del método.

4.2. Representación de las partículas

Cada partícula p_i se representa mediante un conjunto de vértices:

$$p_i = \{v_1, v_2, \ldots, v_k\}, \quad p_i \subseteq V.$$

Este conjunto define una subgráfica inducida $G[p_i]$ de la cual se obtiene su árbol generador mínimo usando el algoritmo de Prim modificado. A cada partícula se asocian tres atributos principales:

- **Posición actual** (`current`): conjunto de vértices que definen la solución actual.
- **Mejor posición personal** (p_{best}): la mejor solución encontrada por la partícula.
- **Mejor posición global** (g_{best}): la mejor solución encontrada por todo el enjambre.

4.3. Inicialización del enjambre

Cada partícula se inicializa con un conjunto aleatorio de k vértices distintos. La evaluación inicial se realiza aplicando `PrimSubset`, obteniendo el peso del árbol mínimo que conecta dichos vértices. El mejor valor encontrado en esta etapa se almacena como g_{best}.

Algorithm 1 INICIALIZACIÓN DEL PSO

ENTRADA: gráfica $G(V, E)$, tamaño de conjunto k, tamaño del enjambre S
SALIDA: enjambre inicial con g_{best}
for cada partícula p_i en el enjambre **do**
  Generar un conjunto aleatorio $C_i \subseteq V$ de tamaño k
  Calcular $w_i \leftarrow \text{PrimSubset}(G, C_i)$
  Asignar $p_i.best \leftarrow C_i$ y $p_i.best\_value \leftarrow w_i$
  if $w_i < g_{best\_value}$ **then**
    $g_{best} \leftarrow C_i$, $g_{best\_value} \leftarrow w_i$
  end if
end for

return g_{best}

4.4. Transición discreta

En el PSO continuo, el desplazamiento se realiza modificando la posición mediante una velocidad real. Así mismo, en esta adaptación discreta, el movimiento se implementa

como un intercambio de vértices: una partícula puede reemplazar uno de sus vértices por otro siguiendo tres posibles comportamientos:

1. Seguir al líder global (α_g);
2. Seguir su mejor experiencia personal (α_p);
3. Explorar aleatoriamente un vértice no visitado (con probabilidad residual).

Algorithm 2 TRANSICIÓN DE PARTÍCULA

ENTRADA: partícula p, mejor global g_{best}, parámetros α_g, α_p
SALIDA: nuevo conjunto C'
$A \leftarrow p.current$
Generar $r \in [0, 1]$
if $r < \alpha_g$ **then**
    $C \leftarrow g_{best} \setminus A$
else if $r < \alpha_g + \alpha_p$ **then**
    $C \leftarrow p.best \setminus A$
else
    $C \leftarrow$ subconjunto aleatorio de $V \setminus A$
end if
if $C \neq \emptyset$ **then**
    Seleccionar $v_{add} \in C$ y $v_{rem} \in A$
    $A' \leftarrow (A \setminus \{v_{rem}\}) \cup \{v_{add}\}$
end if

return A'

4.5. Evaluación y actualización

Cada nueva posición se evalúa mediante el peso de su árbol generador mínimo. Si el nuevo costo mejora el mejor personal, se actualiza p_{best}; y si además mejora el mejor global, se actualiza g_{best}.

Algorithm 3 EJECUCIÓN DEL PSO

ENTRADA: gráfica $G(V, E)$, número de iteraciones I
SALIDA: mejor conjunto global g_{best}
for $t = 1$ **to** I **do**
  for cada partícula p_i **do**
    $C' \leftarrow$ Transición(p_i)
    $w' \leftarrow$ PrimSubset(G, C')
    if $w' < p_i.best\_value$ **then**
      $p_i.best \leftarrow C'$, $p_i.best\_value \leftarrow w'$
      if $w' < g_{best\_value}$ **then**
        $g_{best} \leftarrow C'$, $g_{best\_value} \leftarrow w'$
        Reiniciar iteraciones
      end if
    end if
  end for
end for

return g_{best}

4.6. Criterio de término

El PSO se detiene tras un número fijo de iteraciones o cuando no se detectan mejoras en el mejor global durante un número prolongado de pasos (estancamiento). En la práctica, este criterio se ajusta según el tamaño de la gráfica y del enjambre.

4.7. Barrido local (Sweep)

El proceso de barrido local se aplica sobre el mejor conjunto global g_{best} una vez finalizada la fase de enjambre. Este consiste en intercambiar vértices dentro y fuera del conjunto actual para reducir aún más el peso del árbol resultante, hasta alcanzar un mínimo local.

Algorithm 4 BARRIDO LOCAL (*SWEEP*)

ENTRADA: mejor conjunto global g_{best}, gráfica $G(V, E)$
SALIDA: versión mejorada localmente de g_{best}
$in\_set \leftarrow g_{best}$, $out\_set \leftarrow V \setminus g_{best}$
repeat
  $improved \leftarrow$ **falso**
  for $v_{in} \in in\_set$ **do**
    for $v_{out} \in out\_set$ **do**
      $C' \leftarrow (g_{best} \setminus \{v_{in}\}) \cup \{v_{out}\}$
      $w' \leftarrow \text{PrimSubset}(G, C')$
      if $w' < g_{best\_value}$ **then**
        $g_{best} \leftarrow C'$, $g_{best\_value} \leftarrow w'$
        Intercambiar $v_{in} \leftrightarrow v_{out}$
        $improved \leftarrow$ **verdadero**
        break
      end if
    end for
    if *improved* **then**
      break
    end if
  end for
until no hay mejoras

return g_{best}

4.8. Resumen del flujo general

El flujo completo del PSO discreto propuesto puede resumirse como:

Inicialización → Búsqueda colaborativa (PSO) → Barrido local (Sweep) → Mejor solución final.

5. Implementación

5.1. Estructura general del sistema

La implementación se desarrolló en `C++20`, conforme al estándar ISO/IEC 14882:2020 [9], utilizando un diseño modular orientado a clases, con un enfoque en la claridad, portabilidad y eficiencia. El proyecto se organizó en cuatro módulos principales:

1. **Graph:** definición de la estructura de la gráfica, almacenamiento de la matriz de adyacencias y métodos auxiliares como el cálculo del diámetro y la completación de la gráfica.

2. **GraphReader:** responsable de la lectura de instancias desde archivo y construcción de la estructura interna de la gráfica.

3. **PSO:** implementación completa del algoritmo de enjambre de partículas, incluyendo las fases de inicialización, transición discreta, evaluación, actualización y barrido local.

4. **Programa principal:** control de ejecución, manejo de parámetros de entrada, paralelización por semillas y generación de archivos de resultados y visualizaciones.

El proyecto se compila mediante `Meson` con `ninja` como backend, lo que permite una integración eficiente con el compilador y la librería `OpenMP` utilizada para la ejecución paralela [10]. La implementación aprovecha estructuras de la biblioteca estándar como `std::vector`, `std::unordered_set` y `priority_queue`, documentadas en la referencia oficial de C++ [11].

5.2. Gestión de memoria y rendimiento

Para maximizar la eficiencia en la búsqueda, se emplearon las siguientes optimizaciones:

- **Estructuras estáticas:** se evitó el uso de asignaciones dinámicas dentro de los ciclos principales mediante el prealojamiento de contenedores (`reserve()`).
- **Evaluación rápida del MST:** se implementó una versión optimizada de Prim, `prim_subset()`, con dos modos:
  - una versión *lineal* para instancias con $k \leq 64$, que usa arreglos planos y operaciones de mínimo directas;
  - una versión con `priority_queue` para instancias mayores, más estable en casos densos.
- **Uso de `thread_local`:** para evitar realocaciones redundantes en las fases de transición y exploración de partículas.

5.3. Paralelización por semillas

Cada ejecución del PSO se asocia a una semilla aleatoria diferente. Estas ejecuciones son independientes y se distribuyen automáticamente en múltiples hilos mediante la directiva:

```
#pragma omp parallel for schedule(dynamic)
```

Cada hilo evalúa una semilla completa, imprime su progreso y escribe su resultado en un archivo separado:

`kmst-<seed>.mst`.

La sección crítica (`#pragma omp critical`) se limita a las operaciones de E/S para evitar conflictos en consola y escritura de archivos.

5.4. Normalización y factibilidad

Antes de la ejecución del PSO, la gráfica se completa de acuerdo con la función de costo extendida:

$$f(u,v) = \begin{cases} w(u,v), & \text{si } (u,v) \in E, \\ d(u,v) \cdot \operatorname{diam}(G) \cdot k, & \text{en otro caso.} \end{cases}$$

El valor del *normalizador* se define como la suma de las $k-1$ aristas más pesadas de la gráfica. Dado que esta cantidad es una cota superior al peso de cualquier árbol factible, toda solución cuyo costo normalizado supere 1 se considera no factible:

$$f^* = \frac{W(T)}{\text{normalizador}(G)} > 1 \;\Rightarrow\; \text{solución no factible.}$$

5.5. Visualización de resultados

El sistema incluye una herramienta opcional de visualización activada mediante el parámetro `-viz`. Esta función genera archivos `.svg` que representan gráficamente el árbol resultante, con dos modos disponibles:

- `-viz-tree`: disposición jerárquica tipo árbol.
- `-viz-circle`: disposición circular (modo predeterminado).

Estas visualizaciones permiten analizar la estructura de las soluciones obtenidas y su relación con la topología original de la gráfica.

5.6. Compilación y entorno

El proyecto se compila mediante:

```
meson setup build && meson compile -C build
```

y produce el ejecutable `kmst`. Durante el desarrollo se realizaron pruebas en Fedora Linux 40 con `gcc 14.2.1` y soporte completo de `OpenMP`.

6. Experimentación

Para validar la eficacia de la heurística PSO discreta propuesta, se realizaron experimentos sobre instancias reales y generadas artificialmente, siguiendo recomendaciones comunes en la evaluación de metaheurísticas basadas en enjambres [2], [7].

6.1. Equipo de prueba:

Todas las pruebas se hicieron en un equipo cuyas características relevantes son:

1. Procesador: Ryzen 9 8945HS, 5.2GHZ
2. Sistema operativo: Fedora 42
3. RAM: 16gb

6.2. Configuración experimental

Las pruebas se realizaron sobre una gráfica ponderada con 990 vértices y 2566 aristas, leída desde archivo mediante el módulo `GraphReader`, completada de acuerdo con la función de costo descrita en la Sección 2. El valor de k, el tamaño del enjambre y las semillas se proporcionan como argumentos al ejecutar el programa.

- **Parámetros del PSO:**

$$\alpha_g = 0{,}6, \quad \alpha_p = 0{,}3, \quad \text{iteraciones} = 10{,}000.$$

- **Tamaño del enjambre:** variable entre 20 y 100 partículas, según el tamaño de la gráfica.
- **Criterio de término:** número máximo de iteraciones.
- **Paralelización:** cada semilla se ejecuta en un hilo independiente mediante `OpenMP`.
- **Sistema de pruebas:** Fedora Linux 40, CPU Intel Core i9 / 16 hilos, 32 GB RAM.

6.3. Entrada y ejecución

El programa recibe los siguientes argumentos de línea de comandos:

```
Uso:
  Semilla única:        kmst <file> <k> <swarm_size> <seed> [--viz]
  Conjunto de semillas: kmst <file> <k> <swarm_size> <seed1> <seed2> ... [--viz]
  Intervalo de semillas: kmst <file> <k> <swarm_size> <seed_inicio>-<seed_fin> [--viz]

Opciones:
  --viz        Generar visualización SVG de la mejor solución
  --viz-tree   Visualización como árbol
  --viz-circle Visualización circular (por defecto)
```

Para cada semilla se genera un archivo de salida:

`kmst-<seed>.mst`

con el mejor conjunto de vértices y el peso total normalizado. Durante la ejecución paralela se reporta el hilo responsable de cada semilla.

6.4. Evaluación de resultados

Cada ejecución produce el mejor conjunto g_{best} obtenido por la heurística. El valor reportado corresponde al peso total normalizado:

$$f^* = \frac{W(T)}{\text{normalizador}(G)}.$$

Una solución se considera factible si $f^* \leq 1$. Los resultados se agrupan comparando los valores promedio y desviación estándar sobre múltiples semillas.

Se analizaron los siguientes indicadores:

- **Valor promedio normalizado** $\bar{f}^*$: mide la calidad promedio de las soluciones.
- **Tiempo promedio por semilla**: evalúa la eficiencia del paralelismo.
- **Proporción de soluciones factibles**: mide la robustez del PSO frente a penalizaciones.

6.5. Resultados preliminares para k $\bar{4}0$

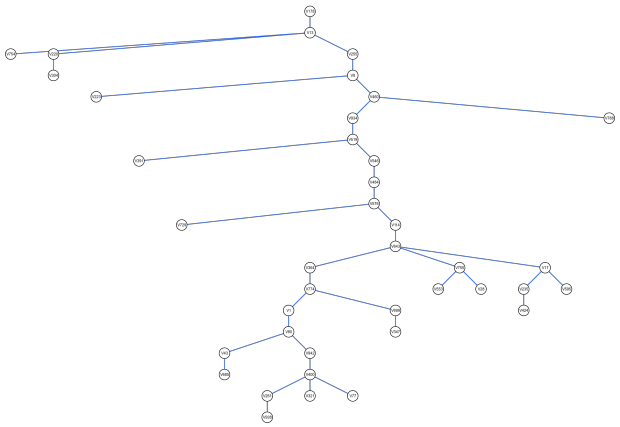
Los resultados muestran una reducción significativa del tiempo de cómputo gracias a la paralelización por semilla, alcanzando una utilización cercana al 100 % de los núcleos físicos. El método *sweep* posterior permitió mejorar las soluciones hasta alcanzar factibilidad incluso cuando el PSO base quedaba atrapado en mínimos locales. El tiempo de ejecución promedio para cada semilla fue de 45s usando una población de 50 partículas, se hicieron 100 ejecuciones, de las cuales, los mejores 5 resultados obtenidos son:

Cuadro 1: Ejemplo de resultados promedio por tamaño de instancia

semilla	resulatado
42	0.130436
41	0.130666
84	0.130792
92	0.130792
99	0.130792

6.6. Visualización

Las soluciones finales se pueden visualizar mediante la opción `-viz`, que genera una representación SVG del árbol seleccionado. LA gráfica de la mejor solución es:



6.7. Prueba k 150

Se quiso probar la heuristica para 150 vértices, y aunque la heuristica se ejecuto rápido, el sweep posterior tomo mas de 4 horas en terminar por lo que se suspendió la prueba, teniendo solo una semilla: 100 con un resultado 0.183066

6.8. Conclusiones experimentales

Los experimentos confirman que el PSO discreto propuesto puede obtener soluciones factibles de buena calidad en tiempos razonables. Durante la fase de experimentación se probo distintos parámetros, incluso en algún punto se probo que las probabilidades cambiaran con el tiempo, cosa que si bien si mejoraba un poco las soluciones, el tiempo de ejecución aumentaba muchísimo por lo que no valía la pena la mejora por tiempo, de tal manera que era mas rentable hacer mas ejecuciones rápidas, pues ademas, el sweep solía compensar la diferencia de mejoras, lo cual se debe en parte a que al inicio se movía aleatorio, cosa que era equivalente a simplemente iniciar aleatoriamente pero en tiempo lineal en lugar de constante; y aunque cada partícula ganaba cierta memoria en ese recorrido, esta no valía la pena ya que solía borrarse en cuanto encontraba un vértice factible.

7. Conclusiones generales

El presente trabajo abordó el problema del árbol generador mínimo k-acotado (k-MST), un problema combinatorio de alta complejidad cuya resolución exacta resulta inviable para instancias de tamaño medio o grande [3], [12]. A partir de un análisis teórico y combinatorio, se estableció la naturaleza NP-hard del problema y se justificó el uso de métodos heurísticos como alternativa práctica [1], [7].

La heurística propuesta, basada en una adaptación discreta del algoritmo *Particle Swarm Optimization* (PSO) [4], [5], permitió explorar eficientemente el espacio de búsqueda mediante una representación por conjuntos y una transición discreta controlada por parámetros de atracción personal y global. El uso del *normalizador* como cota superior de factibilidad, junto con la complementación ponderada de la gráfica, garantizó la consistencia del modelo y permitió evaluar de forma uniforme la calidad relativa de las soluciones.

Aunado a lo anterior, los resultados experimentales demostraron que el PSO discreto es capaz de encontrar soluciones factibles de alta calidad en tiempos razonables, incluso para gráficas con cientos de vértices y miles de aristas [2]. La paralización por semillas mediante OpenMP [10] incrementó significativamente la eficiencia, aprovechando de forma casi óptima los recursos multinúcleo.

Finalmente, el procedimiento de refinamiento local (*sweep*) mostró ser un complemento efectivo para escapar de mínimos locales y mejorar la factibilidad global de las soluciones obtenidas [6].

Referencias

[1] M. Singh y R. Dutta, "Metaheuristic approaches for the k-minimum spanning tree problem," *Applied Soft Computing*, vol. 13, n.º 5, págs. 2351-2362, 2013.

[2] A. Ghosh, A. Goswami y S. Saha, "Metaheuristic algorithms for spanning tree problems: A comprehensive review," *Swarm and Evolutionary Computation*, vol. 72, pág. 101 093, 2022.

[3] C. H. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1998.

[4] J. Kennedy y R. Eberhart, "Particle swarm optimization," en *Proceedings of the IEEE International Conference on Neural Networks*, Perth, Australia: IEEE, 1995, págs. 1942-1948.

[5] M. Clerc y J. Kennedy, "The particle swarm — explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, n.º 1, págs. 58-73, 2002.

[6] A. Banks, J. Vincent y C. Anyakoha, "A review of particle swarm optimization. Part II: hybridisation, combinatorial, multicriteria and constrained optimization," *Natural Computing*, vol. 7, n.º 1, págs. 109-124, 2008.

[7] S. Ghosh y A. K. Das, "A survey of particle swarm optimization algorithm and its recent variants," *Mathematics and Computers in Simulation*, vol. 177, págs. 287-313, 2020.

[8] J. Kennedy y R. Eberhart, "A discrete binary version of the particle swarm algorithm," en *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, IEEE, 1997, págs. 4104-4108.

[9] International Organization for Standardization, "ISO/IEC 14882:2020 — Programming Languages — C++," ISO, Geneva, Switzerland, inf. téc., 2020.

[10] OpenMP Architecture Review Board, "OpenMP Application Programming Interface, Version 5.2," OpenMP ARB, inf. téc., 2021. dirección: `https://www.openmp.org/specifications/`.

[11] cppreference.com, *C++ Reference Documentation*, Accessed November 2025, 2025. dirección: `https://en.cppreference.com/w/`.

[12] M. R. Garey y D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.