Universidad Nacional de San Agustín

Escuela Profesional de Ciencia de la Computación

ROBÓTICA

Bounded Multi-Source Shortest

Presentado por

Ancel Alain Fernando Cruz Chaiña
Gabriel Pacco Huaraca
Julio Enrique Yauri Itucayasi
Juan Manuel Soto Begazo
Fabrizio Miguel Mattos Cahui
Diego Josue Aquino Quispe

Docente

Percy Maldonado Quispe

1. Introducción

El problema de caminos más cortos con una sola fuente (*Single-Source Shortest Path*, SSSP) es fundamental en ciencias de la computación, con aplicaciones en redes de transporte, enrutamiento de datos, planificación de rutas y optimización de sistemas complejos.

Tradicionalmente, el algoritmo de Dijkstra (1959) ha sido la solución estándar para grafos con pesos no negativos, alcanzando una complejidad de $O((m+n)\log n)$ cuando se utilizan estructuras como *heaps de Fibonacci*.

Recientemente, Duan et al. (2025) han propuesto la técnica *Bounded Multi-Source Shortest Path* (BMSSP), un avance teórico que rompe la denominada *sorting barrier* para SSSP en grafos dirigidos con pesos reales no negativos. Esta técnica aprovecha recursión controlada y estructuras de datos especializadas, alcanzando una complejidad de $O(m\log^{2/3} n)$.

2. Algoritmo de Dijkstra

El algoritmo de Dijkstra computa las distancias más cortas desde un nodo origen s hacia todos los demás nodos en un grafo ponderado con aristas no negativas.

Su principio es iterativo: en cada paso selecciona el nodo no visitado con la menor distancia conocida, lo marca como definitivo y relaja las aristas hacia sus vecinos.

2.1. Pseudocódigo

Algorithm 1 Dijkstra(G, s)

```
0: for all v ∈ V do
0:     dist[v] ← ∞
0: end for
0: dist[s] ← 0
0: Q ← cola de prioridad con todos los vértices
0: while Q ≠ ∅ do
0:     u ← extraer_mínimo(Q)
0:     for all vecino v de u do
0:         alt ← dist[u] + peso(u, v)
0:         if alt < dist[v] then
0:             dist[v] ← alt
0:             actualizar(Q, v, dist[v])
0:         end if
0:     end for
0: end while=0
```

3. Bounded Multi-Source Shortest Path (BMSSP)

El algoritmo BMSSP surge como respuesta al obstáculo clásico conocido como la *sorting barrier*, que impide mejorar la complejidad $O(m + n \log n)$ de los algoritmos tradicionales para SSSP en grafos dirigidos con pesos no negativos. Su diseño se basa en la idea de ejecutar búsquedas acotadas desde múltiples fuentes en paralelo, dividiendo el problema en niveles jerárquicos y controlando cuidadosamente el número de vértices explorados en cada paso.

Definición del subproblema

Sea $l \in [0, \lceil \log n/t \rceil]$ el nivel de recursión. En cada nivel se define un conjunto de vértices parciales S (la frontera), cada uno con una cota superior de distancia $\hat{d}[x]$, y un umbral $B > \max_{x \in S} \hat{d}[x]$. La subrutina BMSSP$(l, B, S)$ debe satisfacer:

- Todo vértice incompleto v con $d(v) < B$ alcanza en su camino más corto un vértice completo en S, garantizando conectividad entre las soluciones parciales.
- La salida es un par (B', U), donde B' es un nuevo umbral más ajustado y U es el conjunto de vértices completados con distancias definitivas.

Casos de salida

La recursión puede terminar de dos formas:

1. **Éxito total**: $B' = B$, lo que implica que se completaron todas las distancias estrictamente menores a B.
2. **Progreso parcial**: $B' < B$ y el tamaño de U satisface $|U| = \Theta(k^2 2^{lt})$, lo que asegura que cada llamada al subproblema genera un avance sustancial en el número de vértices procesados, evitando estancamientos.

Complejidad

El análisis de complejidad combina los parámetros de la recursión:

$$O\big((k\,l + t\,l/k + t)\,|U|\big).$$

El equilibrio entre k (controla el tamaño de los subconjuntos) y t (controla la granularidad de los niveles) permite obtener una complejidad global

$$O(m \log^{2/3} n),$$

rompiendo por primera vez la barrera $O(m + n \log n)$ de los algoritmos deterministas clásicos.

Intuición

El funcionamiento puede entenderse en tres pasos:

- **Divide-and-conquer en niveles:** el problema se divide en niveles recursivos, lo que limita la profundidad de la exploración a $\lceil \log n/t \rceil$.
- **Selección de pivotes:** en cada nivel se reduce el tamaño de la frontera a un subconjunto representativo. Estos pivotes actúan como "testigos" de caminos cortos y permiten controlar la expansión de la búsqueda.
- **Procesamiento acotado:** se usan umbrales B para segmentar el espacio de distancias. De este modo, en lugar de procesar todo el grafo de una vez, se asegura que cada llamada avance sólo hasta un límite y luego devuelva vértices completados.

Interpretación de parámetros

- l: nivel de recursión, determina la profundidad actual en la jerarquía de subproblemas.
- t: factor de escalamiento que controla el número de niveles en la división.
- k: controla el tamaño de los conjuntos de pivotes y equilibra el costo de exploración por nivel.
- B: umbral superior de distancias procesadas en la llamada actual; evita trabajo innecesario sobre distancias mayores.

Relevancia práctica

Aunque BMSSP es un avance principalmente teórico, su estrategia de dividir por niveles y acotar expansiones tiene implicaciones prácticas en:

- **Grandes redes dirigidas**, como grafos de transporte urbano o enrutamiento de Internet, donde los caminos más cortos deben calcularse en escalas de millones de nodos.
- **Análisis de redes sociales**, que exhiben estructuras *scale-free* con hubs de alta conectividad, escenario donde BMSSP mostró aceleraciones drásticas en los experimentos.
- **Optimización en paralelo**, dado que las llamadas a BMSSP(l, B, S) en distintos subconjuntos pueden ejecutarse de manera concurrente.

4. Comparación de Complejidad

Algoritmo	Complejidad temporal
Dijkstra (1959)	$O(m + n \log n)$
BMSSP (Duan et al., 2025)	$O(m \log^{2/3} n)$

BMSSP constituye el primer algoritmo determinista que mejora la barrera de $O(m + n \log n)$ para SSSP en grafos dirigidos.

5. Repositorio

La implementación completa de Dijkstra y BMSSP, junto con un generador de grafos aleatorios y scripts de análisis, se encuentra disponible en:

`https://github.com/MJSoto123/robotics-labs/tree/main`

El repositorio está organizado en carpetas de manera modular para facilitar la comprensión y la reproducibilidad de los experimentos:

- **include**/: contiene los archivos de cabecera (`.h`) donde se definen las estructuras de datos y las interfaces de los algoritmos:
  - `bmssp.h`: declaración de la función principal del algoritmo BMSSP.
  - `dijkstra.h`: definición de la versión clásica del algoritmo de Dijkstra utilizada como baseline.
  - `data_structure_d.h`: implementación de las estructuras de datos auxiliares usadas en BMSSP.
  - `graph_generator.h`: interfaz para la generación de grafos aleatorios en distintos modelos.
  - `types.h`: definiciones de tipos básicos y alias para mejorar la legibilidad del código.
- **src**/: contiene las implementaciones en C++ (`.cpp`) de los algoritmos y utilidades:
  - `bmssp.cpp`: núcleo del algoritmo BMSSP, incluyendo la recursión por niveles y el manejo de pivotes.
  - `dijkstra.cpp`: implementación estándar del algoritmo de Dijkstra.
  - `data_structure_d.cpp`: detalles de las estructuras de datos empleadas por BMSSP.

- `graph_generator.cpp`: funciones concretas para crear grafos de distintos modelos (Erdős–Rényi, Barabási–Albert, Watts–Strogatz, Grid 2D, DAG en capas, etc.).

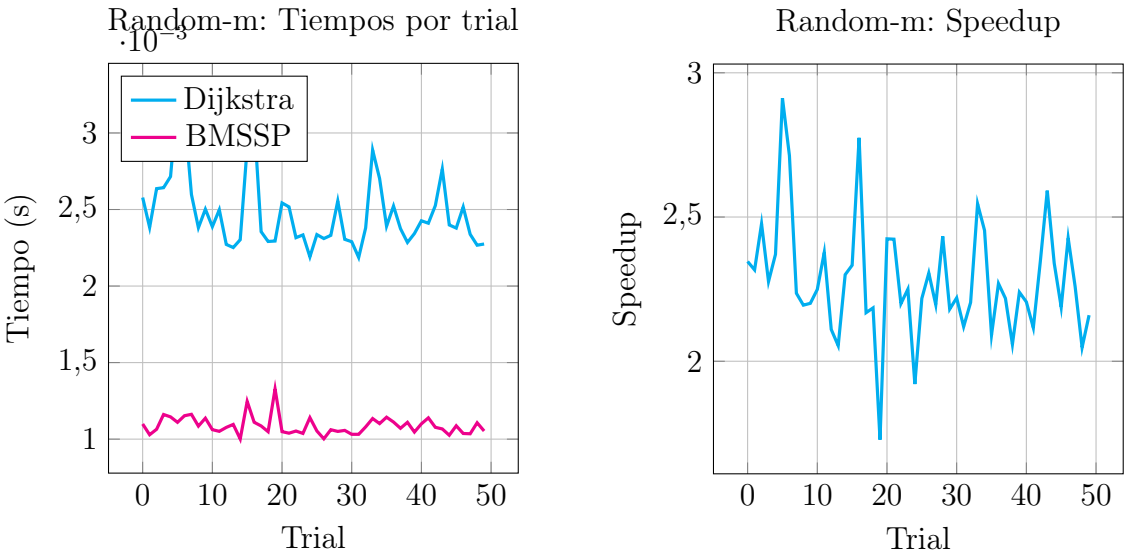
- **test**/: directorio principal para la experimentación y benchmarking:
  - `main.cpp`: programa de entrada que permite ejecutar comparaciones entre Dijkstra y BMSSP, configurando parámetros como número de nodos, número de aristas o tipo de grafo.
  - `benchmark/`: carpeta que almacena los resultados de las corridas experimentales en formato `.csv`. Cada archivo corresponde a un modelo de grafo (`times_random.csv`, `times_er.csv`, `times_ba.csv`, etc.).

Esta organización modular permite compilar los algoritmos de forma independiente, ejecutar experimentos reproducibles y analizar los resultados con facilidad. Además, la carpeta de `benchmark` centraliza la evidencia experimental, lo que facilita la integración de los gráficos de comparación presentados en este informe.

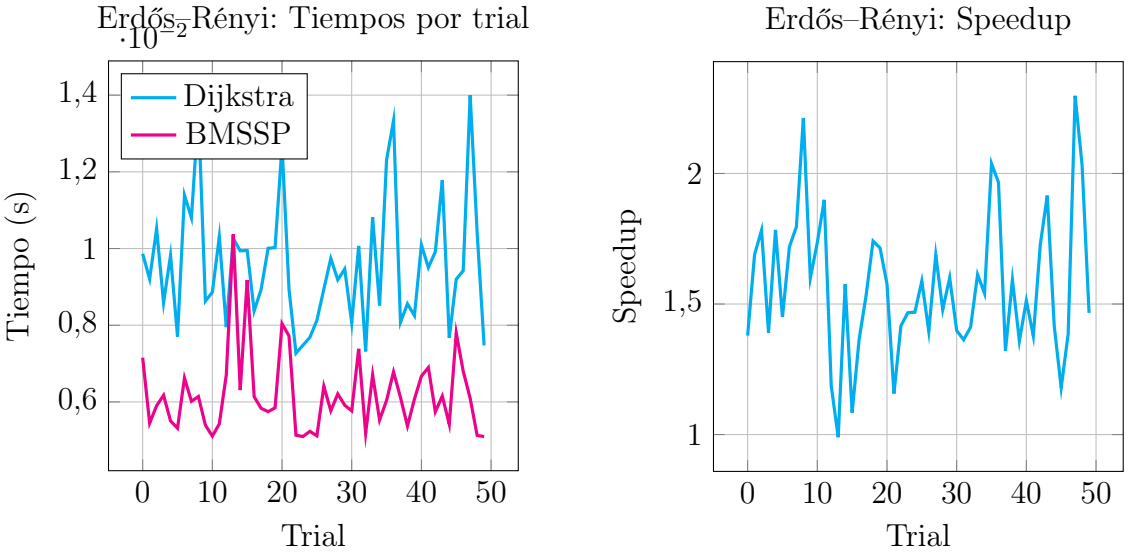
6. Experimentos con Grafos Aleatorios

Se evaluó el desempeño de Dijkstra y BMSSP en seis modelos generativos de grafos, cada uno con propiedades estructurales distintas que impactan directamente en el tiempo de ejecución y el *speedup* relativo $t_{\text{Dijkstra}}/t_{\text{BMSSP}}$.

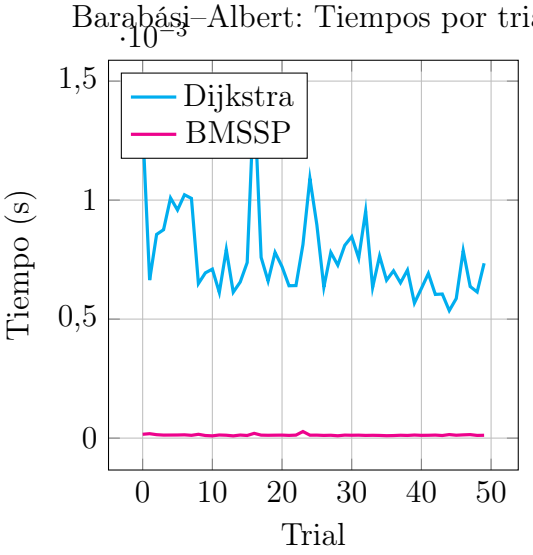
En primer lugar, en el modelo **Random-m**, donde las m aristas se asignan de manera uniforme entre los n vértices, los tiempos de Dijkstra se mantuvieron estables en torno a $2{,}5 \times 10^{-3}$s, mientras que BMSSP se estabilizó cerca de $1{,}0 \times 10^{-3}$s. El *speedup* se sostuvo alrededor de $2\times$, lo que muestra una ventaja moderada pero constante de BMSSP en escenarios sin estructura específica.

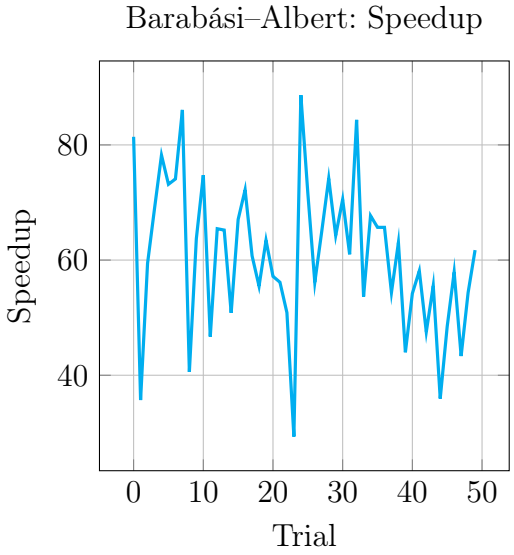


El modelo **Erdős–Rényi**, basado en la adición de aristas con probabilidad p, presentó mayor variabilidad en los tiempos debido a fluctuaciones en la densidad. En este caso, Dijkstra y BMSSP alternaron ventajas locales, aunque en promedio el *speedup* osciló entre $1{,}2\times$ y $2{,}0\times$, confirmando que BMSSP maneja mejor grafos con densidad irregular.

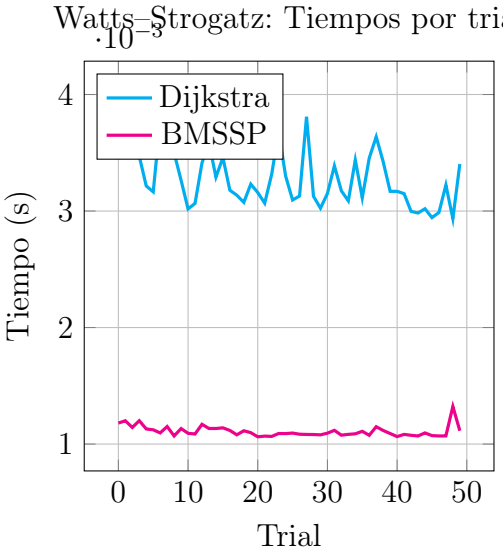


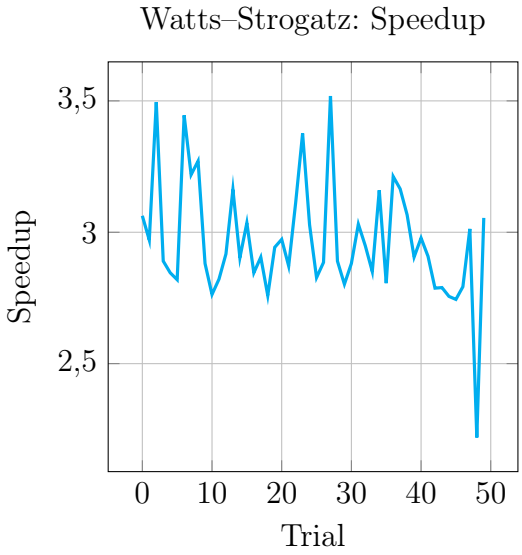
Un caso especialmente favorable para BMSSP fue el modelo **Barabási–Albert**, caracterizado por su distribución *scale-free* con hubs altamente conectados. Allí, Dijkstra se vio penalizado por la necesidad de relajar un gran número de aristas en los nodos hub, mientras que BMSSP aprovechó sus pivotes y fronteras múltiples para reducir drásticamente el trabajo. El resultado fue un *speedup* extraordinario, superior a 60× en promedio y con picos cercanos a 90×, lo que resalta la superioridad de BMSSP en redes con estructura jerárquica desigual.



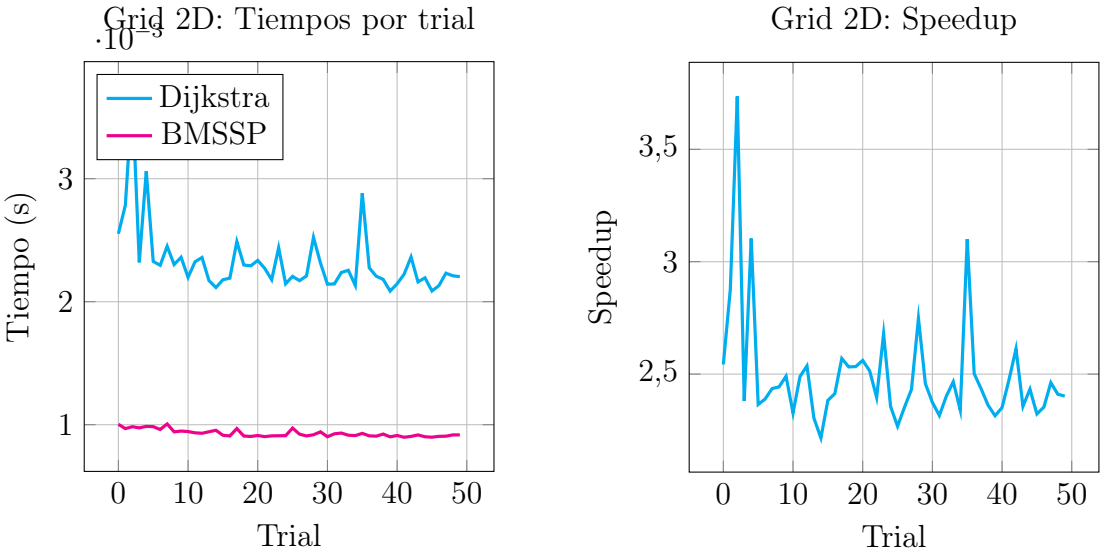


En el modelo **Watts–Strogatz**, representativo de grafos *small-world* con alto clustering local y caminos globales cortos, BMSSP se mantuvo en torno a $1{,}0 \times 10^{-3}$s frente a los $3{,}0 \times 10^{-3}$s de Dijkstra. El *speedup* osciló entre 2,5× y 3,5×, explicable por la presencia de *shortcuts*, que reducen la profundidad de búsqueda y potencian las divisiones en niveles de BMSSP.

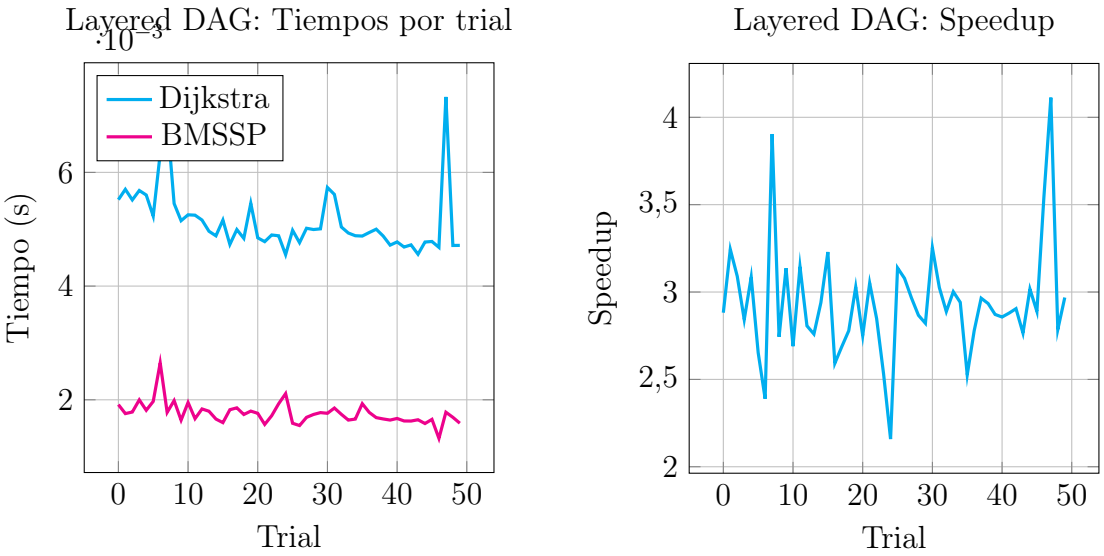




En el caso del **Grid 2D**, un grafo regular en forma de malla donde cada nodo se conecta con sus vecinos inmediatos, ambos algoritmos recorrieron caminos similares. Sin embargo, BMSSP redujo la redundancia en las expansiones capa por capa, alcanzando un *speedup* estable entre 2,3× y 3,0×, frente a los $2{,}5 \times 10^{-3}$s de Dijkstra.



Finalmente, en el **Layered DAG**, un grafo acíclico con estructura en capas, Dijkstra mostró tiempos altos debido a la densidad inter-capa, mientras que BMSSP se mantuvo entre 1,5 y 2,0 $\times 10^{-3}$s. El *speedup* promedio fue cercano a 3×, con picos de 4×, lo que refleja cómo la estrategia de múltiples fuentes se adapta de forma natural a grafos jerárquicos.



En conjunto, los resultados experimentales muestran que BMSSP obtiene mejoras consistentes en todos los modelos, con ventajas modestas en grafos aleatorios homogéneos y mejoras mas significativas en redes con hubs o estructura jerárquica marcada.

7. Conclusiones

Los resultados experimentales permiten extraer varias conclusiones clave sobre el desempeño comparativo de Dijkstra y BMSSP:

1. **Patrones según tipo de grafo:**
   - En grafos homogéneos y sin estructura marcada (Random-m y Erdős–Rényi), la mejora de BMSSP sobre Dijkstra es moderada (entre 1,5× y 2×), ya que ambos algoritmos recorren un número de aristas similar y el beneficio de las fronteras múltiples es limitado.
   - En grafos con estructura regular (Grid 2D y Layered DAG), la ganancia se eleva a la franja de 2,5× a 4×, porque BMSSP evita redundancias en la expansión capa por capa y aprovecha la jerarquía natural de las capas.
   - En grafos con propiedades *small-world* (Watts–Strogatz), los *shortcuts* reducen la profundidad de la exploración, lo que potencia la estrategia recursiva de BMSSP y genera *speedups* entre 2,5× y 3,5×.
   - En grafos heterogéneos con hubs altamente conectados (Barabási–Albert), BMSSP alcanza mejoras espectaculares (hasta 90×), pues evita los cuellos de botella que penalizan a Dijkstra en nodos de altísimo grado.
2. **Relevancia teórica:** BMSSP representa un avance significativo al romper la barrera de *sorting* en SSSP, alcanzando complejidad $O(m \log^{2/3} n)$ frente a los $O(m + n \log n)$ de Dijkstra.
3. **Impacto práctico:** La mejora asintótica es especialmente relevante para grafos de gran escala. Para grafos pequeños o moderados, Dijkstra sigue siendo competitivo y práctico por su sencillez.
4. **Consideraciones de implementación:** El potencial de BMSSP depende de un diseño cuidadoso de estructuras de datos y de un manejo eficiente de las fronteras y pivotes, lo que supone un reto adicional respecto a Dijkstra.
5. **Confirmación experimental:** Los experimentos muestran que BMSSP obtiene mejoras consistentes en todos los escenarios y que la magnitud de la aceleración depende fuertemente de la estructura del grafo: cuanto mayor es la heterogeneidad en la conectividad o la jerarquía en las capas, mayor es la ventaja práctica frente a Dijkstra.

Referencias

[1] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959.

[2] Ran Duan, Jiayi Mao, Xiao Mao, Xinkai Shu, Longhui Yin. Breaking the Sorting Barrier for Directed Single-Source Shortest Paths. *arXiv preprint arXiv:2504.17033*, 2025.