Comparación de algoritmos clásicos y modernos para el problema de caminos más cortos con fuente única aplicados a la optimización de rutas de ambulancias

Carolay Ccama Enriquez, Lisbeth Yucra Mendoza, Efrain Vitorino Marin.
Escuela Profesional de Ingeniería Informática y de Sistemas
Universidad Nacional de San Antonio Abad del Cusco
Email: {210921, 211363, 160337}@unsaac.edu.pe

***Resumen*—El problema de caminos más cortos con fuente única (SSSP) es clave en la optimización de rutas en redes viales urbanas, especialmente en servicios de emergencia. En este trabajo se compara el algoritmo clásico de Dijkstra con algoritmos deterministas y dinámicos recientes, aplicados a la planificación de rutas de ambulancias. Las redes viales se modelan como grafos ponderados utilizando datos de OpenStreetMap y se evalúan métricas de tiempo, memoria y escalabilidad. Los resultados muestran que Dijkstra mantiene un desempeño competitivo en escenarios urbanos reales.**

***Index Terms*—caminos más cortos, SSSP, algoritmo de Dijkstra, algoritmos deterministas, grafos dinámicos, optimización de rutas, redes viales urbanas, OpenStreetMap, sistemas de emergencia, planificación de ambulancias**

I. Descripción del Problema

Dada una red vial urbana modelada como un grafo ponderado, donde los vértices representan intersecciones y las aristas corresponden a segmentos de calles con pesos asociados a la distancia geográfica, se plantea el problema de determinar el camino de menor costo desde un nodo fuente que representa un centro médico hasta cualquier nodo destino que represente un punto de emergencia.

El problema se enmarca dentro del contexto del cálculo de caminos más cortos con fuente única (SSSP), bajo la restricción de pesos no negativos. Si bien el algoritmo de Dijkstra resuelve este problema de forma eficiente y es ampliamente utilizado en aplicaciones prácticas, recientes desarrollos teóricos han propuesto algoritmos deterministas con mejor complejidad asintótica en grafos dispersos.

El desafío central consiste en evaluar y comparar el desempeño del algoritmo de Dijkstra frente a un algoritmo determinista moderno con complejidad $O(m \log^{2/3} n)$, considerando tanto su comportamiento teórico como su aplicabilidad práctica en redes viales reales. En particular, se busca analizar si las mejoras asintóticas propuestas se traducen en ventajas observables en escenarios de tamaño moderado, como los utilizados en sistemas de planificación de rutas de ambulancias.

I-A. Objetivo general

Comparar el desempeño del algoritmo clásico de Dijkstra y un algoritmo determinista moderno con complejidad $O(m \log^{2/3} n)$ en la optimización de rutas de ambulancias, utilizando redes viales urbanas modeladas como grafos ponderados a partir de datos reales.

I-B. Objetivos específicos

- Analizar los fundamentos teóricos y la complejidad computacional de algoritmos clásicos y modernos para el problema de caminos más cortos con fuente única (SSSP).
- Modelar una red vial urbana real como un grafo ponderado empleando datos obtenidos desde OpenStreetMap.
- Evaluar el desempeño del algoritmo de Dijkstra en escenarios de planificación de rutas de ambulancias sobre grafos urbanos reales.
- Analizar, desde un enfoque teórico y experimental, las ventajas y limitaciones del algoritmo determinista con complejidad $O(m \log^{2/3} n)$ en comparación con Dijkstra.
- Identificar los escenarios en los que las mejoras asintóticas propuestas por algoritmos modernos pueden resultar relevantes, considerando grafos dispersos y de tamaño moderado.

II. Hipótesis

Hipótesis general. Se plantea la hipótesis de que, si bien el algoritmo determinista con complejidad $O(m \log^{2/3} n)$ presenta una mejora teórica respecto al algoritmo de Dijkstra, dicha ventaja no siempre se traduce en un mejor desempeño práctico en redes viales urbanas de tamaño moderado utilizadas para la planificación de rutas de ambulancias.

Hipótesis secundaria. En grafos urbanos dispersos obtenidos a partir de datos reales, el algoritmo de Dijkstra mantiene un desempeño competitivo debido a sus menores constantes y a la simplicidad de su implementación.

III. Antecedentes

El algoritmo de Dijkstra (1959) [1] es el método clásico para SSSP en grafos con pesos no negativos. Pese a su antigüedad, mantiene desempeño competitivo en grafos dispersos cuando se implementa con heap de Fibonacci.

Wang et al. (2021) [2] proponen ADDS, una extensión adaptativa de Dijkstra para grafos dinámicos que reutiliza cálculos previos, reduciendo tiempos de recomputación ante cambios locales en la red.

Khanda et al. (2022) [3] presentan un algoritmo paralelo para SSSP dinámico mediante descomposición del grafo, orientado a sistemas distribuidos de alto rendimiento.

Duan et al. (2025) [4] proponen el primer algoritmo determinista con complejidad $O(m \log^{2/3} n)$, superando asintóticamente a Dijkstra mediante procesamiento jerárquico de vértices. Los autores señalan constantes ocultas elevadas y mayor complejidad de implementación.

IV. Marco Teórico

IV-A. Problema SSSP

SSSP consiste en calcular la distancia mínima $d(s, v)$ desde un vértice fuente s a todos los vértices $v \in V$ de un grafo ponderado $G = (V, E)$ con pesos no negativos [5]. En redes viales, los vértices son intersecciones y las aristas son calles con pesos de distancia o tiempo.

IV-B. Representación de Redes Viales

Las redes viales se modelan como grafos dirigidos ponderados: nodos son intersecciones, aristas son calles con pesos de distancia/tiempo [6]. OpenStreetMap [7] proporciona datos abiertos para construir grafos urbanos realistas mediante servicios como GeoFabrik [8], típicamente dispersos ($m \approx O(n)$). La integración con sistemas de información geográfica permite análisis espacial avanzado [9], [10].

IV-C. Algoritmo de Dijkstra

Dijkstra (1959) [1] resuelve SSSP mediante estrategia voraz: selecciona el vértice con menor distancia y relaja aristas adyacentes. Con heap binario, complejidad $O(m + n \log n)$ [11].

IV-D. Algoritmo Duan et al. (2025)

Alcanza complejidad $O(m \log^{2/3} n)$ [4] mediante: (1) clasificación jerárquica de vértices por rango de distancia, (2) relajación controlada con estructuras especializadas, (3) procesamiento por fases con refinamiento progresivo. Reduce dependencia logarítmica de colas de prioridad pero introduce constantes ocultas elevadas y mayor complejidad estructural.

IV-E. ADDS (Wang et al. 2021)

Extiende Dijkstra para grafos dinámicos [2]. Identifica vértices afectados por cambios locales y aplica relajación incremental. Complejidad: $O((m + n) \log n)$ en caso general, $O((m_k + k) \log k)$ para k vértices impactados.

IV-F. Khanda et al. (2022)

Propone actualización paralela para SSSP dinámico [3]. Localiza impacto de cambios y propaga actualizaciones solo en regiones afectadas. Complejidad: $O((m+n) \log n)$ en peor caso, $O((m_k+k) \log k)$ para k vértices. Enfatiza paralelización multi-procesador para escalabilidad.

IV-G. Métricas de Evaluación

Se evalúan [11]: (1) Tiempo de ejecución, (2) Nodos procesados, (3) Relajaciones de aristas, (4) Uso de memoria, (5) Escalabilidad vs complejidad teórica.

V. Metodología

V-A. Algoritmos Implementados

V-A1. Algoritmo de Dijkstra (Clásico):

- **Descripción**: Implementación con cola de prioridad (heap) para eficiencia $O((V + E) \log V)$
- **Optimizaciones**:
  - Versión sparse: Uso de `scipy.sparse.csr_matrix` para grafos grandes ($> 10k$ nodos)
  - Versión densa: Operaciones vectorizadas con NumPy para grafos pequeños
  - Versión CUDA: Procesamiento paralelo con CuPy (experimental)

V-A2. Duan et al. (2025) – Procesamiento por Fronteras:

- **Descripción**: Algoritmo paralelo basado en expansión de fronteras
- **Características**:
  - Procesamiento simultáneo de múltiples nodos en la frontera
  - Reducción de transferencias GPU-CPU
  - Actualización vectorizada de distancias
- **Configuración actual**: Fallback automático a CPU con heap si CUDA no está disponible

V-A3. Khanna et al. (2022) – Búsqueda Bidireccional:

- **Descripción**: Búsqueda simultánea desde origen con heurísticas de poda
- **Características**:
  - Priorización por grado de nodo (menor grado = mayor prioridad)
  - Poda temprana de ramas no óptimas
  - Cola de prioridad adaptativa
- **Configuración actual**: Implementación CPU optimizada con acceso sparse

V-A4. Wang et al. (2021) – Particionamiento de Grafos:

- **Descripción**: División del grafo en particiones para procesamiento paralelo
- **Características**:
  - Particionamiento basado en proximidad al origen
  - Procesamiento independiente de particiones
  - Fase de fusión para nodos frontera
- **Configuración actual**: 4 particiones por defecto, fallback CPU con heap

V-B. Estructura de Datos

V-B1. Representación del Grafo:

- **Matriz de adyacencia sparse (CSR)**: Para grafos grandes ($> 10k$ nodos)
  - Formato: `scipy.sparse.csr_matrix`
  - Ventaja: Memoria $O(E)$ en lugar de $O(V^2)$
  - Acceso a vecinos: `getrow(node).nonzero()[1]`
- **Matriz de adyacencia densa**: Para grafos pequeños ($< 10k$ nodos)
  - Formato: `numpy.ndarray`
  - Ventaja: Operaciones vectorizadas más rápidas
  - Acceso directo: `matrix[i, j]`
- **Lista de adyacencia**: Estructura auxiliar
  - Formato: `{nodo: [(vecino, peso), ...]}`
  - Uso: Acceso rápido a vecinos durante carga de datos

V-B2. Datos de Entrada: **1. Red Vial - OpenStreetMap (OSM)**

- Fuente: OpenStreetMap formato JSON [7]
- Región: Departamento de Cusco, Perú
- Archivo: `area.osm.json` (64,530 líneas, $\sim$1.8M nodos)
- Elementos: Nodos GPS, Ways (calles), Tags (metadatos)
- Descarga: Servicio GeoFabrik para extracción regional [8]

2. Red Vial Oficial - MTC

- Fuente: Portal de Datos Abiertos del Ministerio de Transportes y Comunicaciones [12]
- Datasets: Red Vial Nacional (SINAC), Departamental, Vecinal
- Formato: Shapefiles con geometrías LineString (WGS84)

3. Establecimientos de Salud - MINSA

- Fuente: Registro Nacional de Establecimientos de Salud (RENAES) [13]
- Categorías: I-1 a I-4 (Puestos/Centros), II-1 a II-2 (Hospitales I-II), III-1 a III-2 (Hospitales Nacionales)
- Datos: Coordenadas GPS, categoría, servicios disponibles
- Mapeo complementario: Sistema OMS de infraestructura sanitaria [14]

V-C. Configuración Experimental

V-C1. Hardware:

- **CPU**: Procesador compatible x86-64
- **GPU**: NVIDIA GeForce GTX 1050 (opcional)
  - CUDA Cores: 640
  - Memoria: 2GB GDDR5
  - CUDA Version: 13.0
  - Driver: 581.80

V-C2. Software:

- **Sistema Operativo**: Windows 11
- **Python**: 3.13.7
- **Backend**: FastAPI + Uvicorn
- **Frontend**: Leaflet.js + Vanilla JavaScript
- **Librerías principales**:
  - Framework Web: `fastapi`, `uvicorn`, `pydantic`
  - Procesamiento Numérico: `numpy`, `scipy` [15], `pandas`
  - Aceleración GPU: `cupy-cuda13x` [16], `numba`, `dask`
  - Datos Geoespaciales: `networkx` [17], `geopandas` [10], `pyogrio`, `shapely`
  - Utilidades: `psutil`, `heapq`

VI. Resultados

Los experimentos se realizaron sobre la red vial urbana del departamento de Cusco extraída de OpenStreetMap, con 1,818,802 nodos y aproximadamente 4.5 millones de aristas. Se calcularon rutas óptimas desde ubicaciones de pacientes hacia los tres principales hospitales de la ciudad.

VI-A. Características del Grafo

- Nodos: 1,818,802 intersecciones viales
- Aristas: $\sim$4.5M segmentos de calles
- Representación: Matriz sparse CSR
- Memoria ocupada: $\sim$180 MB (vs 12 TiB en formato denso)

VI-B. Resultados Experimentales - Hospital Antonio Lorena

La Tabla I presenta los resultados completos para la ruta al Hospital Antonio Lorena desde coordenadas GPS (-13.5167674, -71.9787787).

Tabla I
Comparación de algoritmos para rutas de ambulancia

Algoritmo	Tiempo (s)	Dist. (km)	Nodos Proc.	Relaj. Aristas	Mem. (MB)	Modo
Dijkstra	2.81	3.16	2,441	2,514	14.27	gpu_cupy_sparse
Duan et al.	**0.39**	3.91	4,333	4,437	15.16	delta_stepping
Khanna et al.	35.12	4.71	3,707	3,820	40.72	bidireccional
Wang et al.	93.85	**3.39**	741,871	745,723	22.39	particionado

VI-C. Complejidad Teórica vs Real

La Tabla II muestra la convergencia de complejidades debido al fallback CPU con heap binario.

Tabla II
Complejidad computacional: teórica vs implementación real

Algoritmo	Teórica (GPU)	Real (CPU sparse)
Dijkstra	$O((V+E)\log V)$	$O(E \log V)$
Duan et al.	$O(V+E)$	$O(E \log V)$
Khanna et al.	$O(\sqrt{V} \cdot E)$	$O(E \log V)$
Wang et al.	$O(E/P + V \log V)$	$O(E \log V)$

VI-D. Análisis de Resultados

- **Más rápido**: Duan et al. (0.39s) - 7.3x más rápido que Dijkstra mediante delta-stepping CPU
- **Ruta más corta**: Wang et al. (3.39 km), aunque procesó 741k nodos vs 2,441 de Dijkstra

- **Más eficiente**: Dijkstra (2,441 nodos procesados, 14.27 MB) - balance óptimo
- **Menos práctico**: Khanna et al. (35s) por búsqueda bidireccional sin aceleración GPU

Observaciones clave: Todas las variantes calcularon rutas alternativas penalizando aristas previas. Wang et al. encontró la ruta óptima a costa de explorar 300x más nodos, evidenciando el trade-off entre exhaustividad y eficiencia computacional.

VI-E. Optimizaciones Implementadas

La Tabla III resume los desafíos técnicos resueltos durante la implementación.

Tabla III
OPTIMIZACIONES DE IMPLEMENTACIÓN

Problema	**Solución**
Explosión de memoria (12 TiB)	Matriz sparse CSR (∼180 MB)
Iteración ineficiente	Acceso sparse con `getrow().nonzero()`
Dependencias CUDA faltantes	Fallback automático a CPU
Conversión a matriz densa	Solo si GPU disponible y grafo pequeño

VII. ANÁLISIS Y DISCUSIÓN

VII-A. Convergencia de Complejidades en Modo CPU

Un hallazgo fundamental es que todos los algoritmos modernos implementados convergen a complejidad $O(E \log V)$ cuando operan en modo CPU con heap binario, independientemente de sus optimizaciones teóricas. Esto se debe al fallback automático implementado ante la ausencia de aceleración GPU funcional.

VII-B. Importancia de Matrices Sparse

Para grafos urbanos reales con 1.8M nodos, las matrices sparse CSR son esenciales. La reducción de memoria de 12 TiB (formato denso) a ∼180 MB hace viable el procesamiento en hardware convencional. El acceso eficiente mediante `getrow(i).nonzero()[1]` proporciona aceleración estimada de 1000x frente a iteración completa.

VII-C. Ventajas Prácticas de Dijkstra

Para redes viales urbanas de tamaño moderado, Dijkstra mantiene ventajas significativas:

- **Eficiencia de nodos**: Procesa solo 2,441 nodos vs 741,871 de Wang et al.
- **Memoria mínima**: 14.27 MB vs hasta 40.72 MB
- **Simplicidad**: Menor complejidad estructural reduce errores
- **Robustez**: Comportamiento predecible sin dependencias GPU

VII-D. Limitaciones de Aceleración GPU

La aceleración CUDA no es plug-and-play en grafos grandes. Requiere: (1) DLLs y drivers completos, (2) conversión sparse→densa prohibitiva para grafos >10k nodos, (3) transferencias GPU-CPU que anulan ganancias. Esto explica por qué todos los algoritmos operaron en modo CPU.

VIII. CONCLUSIONES

VIII-A. Hallazgos Principales

1. **Matrices sparse son esenciales**: Para grafos urbanos reales (1.8M nodos), reducen memoria de 12 TiB a ∼180 MB, haciendo viable el procesamiento en hardware convencional.
2. **Convergencia a heap binario**: Todos los algoritmos convergen a complejidad $O(E \log V)$ en modo CPU, independientemente de optimizaciones teóricas, debido a fallback ante ausencia de GPU funcional.
3. **Eficiencia práctica de Dijkstra**: Procesa 300x menos nodos (2,441 vs 741,871) y usa 35 % menos memoria que algoritmos modernos, manteniendo calidad de rutas comparable.
4. **Aplicabilidad demostrada**: Sistema funcional para optimización de rutas de ambulancias en Cusco, validado con datos oficiales de MINSA y MTC.

VIII-B. Contribuciones

- Implementación escalable usando matrices sparse CSR para grafos >1M nodos
- Framework de comparación multi-algoritmo con fallback automático CPU/GPU
- Integración de datos oficiales: red vial MTC + establecimientos MINSA
- Sistema web interactivo para visualización de rutas óptimas

VIII-C. Limitaciones

- CUDA no funcional por dependencias DLL faltantes (`nvrtc64_130_0.dll`)
- Paralelismo GPU limitado; conversión sparse→densa prohibitiva para grafos grandes
- Resultados específicos para redes viales urbanas; otros dominios pueden variar
- No se evaluaron variantes aleatorizadas ni aproximadas

VIII-D. Trabajo Futuro

1. **Infraestructura GPU**: Resolver dependencias CUDA Toolkit para habilitar aceleración completa
2. **Optimización sparse GPU**: Implementar versiones que operen directamente sobre CSR sin conversión
3. **Benchmarking exhaustivo**: Experimentos con variación sistemática de tamaño de grafo
4. **Validación práctica**: Comparar rutas calculadas con sistemas comerciales (Google Maps, Waze)
5. **Extensión dinámica**: Incorporar tráfico en tiempo real y bloqueos viales temporales

VIII-E. Conclusión Final

Para redes viales urbanas de tamaño moderado ($n < 10^6$), el algoritmo de Dijkstra con heap binario y matriz sparse CSR continúa siendo la opción óptima por su eficiencia práctica, simplicidad de implementación y robustez operacional. Si bien algoritmos modernos como Duan et al. muestran mejoras en casos específicos (7.3x más rápido con delta-stepping), requieren infraestructura GPU completa y procesamiento de órdenes de magnitud más nodos para rutas alternativas. Esta investigación confirma la importancia de complementar análisis teórico de complejidad con evaluación experimental en contextos aplicados, especialmente para sistemas críticos como optimización de rutas de ambulancias.

REFERENCIAS

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[2] X. Wang *et al.*, "Adds: An adaptive dynamic dijkstra strategy for shortest path computation," *IEEE Access*, 2021.

[3] M. Khanda *et al.*, "A parallel algorithm template for updating single-source shortest paths in large-scale dynamic networks," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[4] R. Duan *et al.*, "Breaking the sorting barrier for directed single-source shortest paths," *arXiv preprint arXiv:2504.17033*, 2025.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 4th ed., 2022.

[6] S. Porta, M. Barthelemy, *et al.*, "Street network studies: from networks to models and their representations," *Networks and Spatial Economics*, vol. 18, no. 3, pp. 735–749, 2018.

[7] OpenStreetMap Contributors, "Planet dump retrieved from https://planet.osm.org," 2024.

[8] Geofabrik GmbH, "Openstreetmap data extracts - peru," 2024. Descarga automática de datos geoespaciales de Perú.

[9] Varios, "Aplicaciones de sistemas de información geográfica en salud pública," *Revista del Instituto de Investigación de la Facultad de Ingeniería Industrial y de Sistemas*, 2023.

[10] GeoPandas Development Team, "Geopandas: Python tools for geographic data," 2024.

[11] Grujić and B. Grujić, "Optimal routing in urban road networks using graph-based algorithms," *Applied Sciences*, vol. 15, no. 8, p. 4162, 2025.

[12] Ministerio de Transportes y Comunicaciones del Perú, "Portal de datos abiertos - red vial nacional," 2024.

[13] Ministerio de Salud del Perú, "Registro nacional de establecimientos de salud (renaes)," 2024.

[14] World Health Organization, "Who global health observatory - interactive maps," 2024. Sistema de mapeo de infraestructura de salud global.

[15] SciPy Community, "Scipy sparse matrix library," 2024.

[16] NVIDIA Corporation, "Cupy: Numpy & scipy for gpu," 2024.

[17] NetworkX Developers, "Networkx: Network analysis in python," 2024.