Análisis de algoritmos Dijkstra vs A*

1st André Chávez Contreras
Universidad de Xalapa
Complejidad algorítmica
Xalapa, México
ux23ii263@ux.edu.mx

***Abstract*—This report presentes a comparative analysis of the Dijkstra and A* algorithms, both algorithms were implemented to find the shortest path between nodes in a graph that represents the city of Xalapa Veracruz Mexico, the study evaluates the behavior of the algorithms in terms of iterations, shortest paths and time.**

***Index Terms*—Sorting algorithms, Bubble Sort, Quick Sort.**

I. Introducción

Los algoritmos de búsqueda de caminos o nodos son fundamentales para distintas aplicaciones y objetivos, desde navegadores hasta aplicaciones en robótica o el enrutamiento de redes, dentro de los algoritmos más destacados se encuentran Dijkstra y A*, los cuales son óptimos para encontrar los mejores caminos en grafos de gran tamaño, el objetivo de este proyecto es probar los dos algoritmos en la ciudad de Xalapa Veracruz, para poder determinar las ventajas y desventajas de cada algoritmo.

A. ¿Cuál es el algoritmo más eficiente para encontrar caminos?

El propósito de conocer el algoritmo más eficiente para encontrar caminos cortos es poder conocer las fortalezas y debilidades de cada uno de los algoritmos, y poder ver cuál se puede utilizar para distintos enfoques.

B. Objetivos

- Conocer el algoritmo más óptimo para grafos de mayor tamaño.
- Encontrar el algoritmo menos costoso en recursos de hardware.
- Determinar cuál es el algoritmo más preciso.

C. Justificación

Comparar los algoritmos Dijktra y A* en una red urbana real proporciona un análisis valioso y realista sobre cómo trabajan estos algoritmos en situaciones reales en lugar de ambientes teóricos o simplificados.

II. Estado del arte

A. Algoritmo de búsqueda

Un algoritmo de búsqueda es una técnica computacional usada para localizar un elemento o conjunto de elementos en estructuras de datos como listas o grafos. Los algoritmos de búsqueda se clasifican en informados y no informados, según usen o no heurísticas, y son esenciales en la inteligencia artificial, el análisis de datos y la optimización de rutas [1].

B. Algoritmo de Dijkstra

Dijkstra es un algoritmo de búsqueda que encuentra el camino más corto entre nodos en un grafo con pesos positivos. Funciona calculando las rutas de menor costo posibles desde el nodo inicial hacia todos los otros nodos del grafo, y garantiza hallar la solución óptima en redes de baja densidad o cuando se requiere máxima precisión. Su complejidad es $O(V^2)$ o $O((V+E) \log V)$ [2].

C. Algoritmo A*

El algoritmo A* es un método de búsqueda informada que combina el costo acumulado desde el origen y una heurística que estima la distancia restante al destino. Con una buena heurística, reduce significativamente el número de nodos explorados respecto a Dijkstra, lo que lo hace más rápido en aplicaciones de gran escala y en tiempo real. Su eficiencia depende de la calidad de la heurística elegida, con una complejidad promedio de $O((V+E) \log V)$ [3].

III. Metodología

A. Codificación de los algoritmos

Para desarrollar y comparar los algoritmos de Dijkstra y A*, se implementaron en Python, debido a su flexibilidad y capacidad para manejar estructuras de datos como grafos y colas de prioridad. Los algoritmos fueron codificados en funciones modulares que reciben como entrada un grafo representado mediante un diccionario o una matriz de adyacencia. Cada algoritmo realiza los cálculos correspondientes y retorna el camino más corto entre el nodo inicial y el nodo destino.

B. Pseudocódigos

0: **Entrada:** Grafo $G = (V, E)$, nodo de inicio s
0: **Salida:** Distancias mínimas de s a todos los nodos de V
0: **for** cada nodo $v \in V$ **do**
0: $dist[v] \leftarrow \infty$ {Inicializar distancia infinita}
0: $prev[v] \leftarrow$ null {Predecesor desconocido}
0: **end for**
0: $dist[s] \leftarrow 0$ {La distancia del nodo de inicio es 0}
0: $Q \leftarrow V$ {Conjunto de nodos no procesados}
0: **while** $Q \neq \emptyset$ **do**
0: $u \leftarrow$ nodo en Q con la distancia mínima $dist[u]$
0: $Q \leftarrow Q \setminus \{u\}$ {Eliminar u de Q}
0: **for** cada vecino v de u **do**
0: **if** $v \in Q$ **then**
0: $alt \leftarrow dist[u] + \text{peso}(u, v)$

0: **if** $alt < dist[v]$ **then**
0: $dist[v] \leftarrow alt$
0: $prev[v] \leftarrow u$
0: **end if**
0: **end if**
0: **end for**
0: **end while**
0: **Retornar** $dist, prev$ =0
0: **Entrada:** Grafo $G = (V, E)$, nodo de inicio s, nodo de destino t, función heurística h
0: **Salida:** El camino más corto de s a t
0: **for** cada nodo $v \in V$ **do**
0: $g[v] \leftarrow \infty$ {Inicializar costo acumulado}
0: $f[v] \leftarrow \infty$ {Inicializar estimación total}
0: $prev[v] \leftarrow$ null {Predecesor desconocido}
0: **end for**
0: $g[s] \leftarrow 0$ {El costo de inicio es 0}
0: $f[s] \leftarrow h(s, t)$ {El valor inicial de la heurística}
0: $openSet \leftarrow \{s\}$ {Conjunto de nodos a explorar}
0: **while** $openSet \neq \emptyset$ **do**
0: $current \leftarrow$ nodo en $openSet$ valor menor $f[current]$
0: **if** $current = t$ **then**
0: **Retornar** el camino desde s a t
0: **end if**
0: $openSet \leftarrow openSet \setminus \{current\}$ {Eliminar $current$ de openSet}
0: **for** cada vecino v de $current$ **do**
0: $tentative_g \leftarrow g[current] +$ peso$(current, v)$
0: **if** $tentative_g < g[v]$ **then**
0: $prev[v] \leftarrow current$
0: $g[v] \leftarrow tentative_g$
0: $f[v] \leftarrow g[v] + h(v, t)$
0: **if** $v \notin openSet$ **then**
0: $openSet \leftarrow openSet \cup \{v\}$
0: **end if**
0: **end if**
0: **end for**
0: **end while**
0: **Retornar** fallo, no hay camino =0

C. Implementación de los algoritmos en el grafo de Xalapa, Veracruz

El grafo utilizado para la implementación de ambos algoritmos corresponde a la ciudad de Xalapa, Veracruz, representada mediante una red de calles y avenidas. Los nodos del grafo representan intersecciones o puntos clave dentro de la ciudad, mientras que las aristas reflejan las conexiones entre estas intersecciones, con pesos asignados según la distancia o el tiempo estimado de recorrido entre los puntos. Para cada uno de los algoritmos, se establecieron rutas entre diferentes puntos de la ciudad de manera aleatoria y se midió el tiempo de ejecución, la calidad de la ruta obtenida y el número de nodos explorados.

IV. Resultados

Se ejecutaron los algoritmos para distintos puntos de la ciudad y se guardaron en la siguiente tabla, la cual muestra la distancia que encontró el algoritmo entre dos puntos aleatorios, la velocidad, el tiempo y la cantidad de iteraciones que realizó.

Algoritmo	Distancia	Velocidad	Tiempo	Iteraciones
Dijkstra	6.6448	40.83	9.76	10348
A*	6.7185	40.27	10.01	2308
Dijkstra	9.3838	40.00	14.08	10774
A*	9.4466	40.00	14.17	1814
Dijkstra	7.0428	42.17	10.02	7623
A*	7.0699	42.21	10.05	2155

Resultados de Dijkstra y A*

A. Distancia

En general, las distancias calculadas por ambos algoritmos fueron bastante similares, con Dijkstra alcanzando una distancia promedio de 6.64, 9.38 y 7.04 en los tres intentos, mientras que A* obtuvo distancias de 6.71, 9.45 y 7.07. Esto sugiere que ambos algoritmos son capaces de encontrar caminos relativamente cercanos en términos de distancia, aunque A* muestra una ligera tendencia a obtener distancias ligeramente mayores.

B. Velocidad

La velocidad promedio se mantuvo constante entre los intentos, con ambos algoritmos alcanzando valores similares en todas las pruebas. Dijkstra mantuvo una velocidad promedio de alrededor de 40 km/h, mientras que A* presentó una pequeña variación en los valores, oscilando entre 40.27 y 42.21 km/h. Estas diferencias en velocidad promedio son mínimas y no parecen afectar significativamente la eficacia de los algoritmos.

C. Tiempo

El tiempo total de ejecución mostró una tendencia consistente en ambos algoritmos, con Dijkstra tomando tiempos de 9.76, 14.08 y 10.02 segundos, mientras que A* tomó 10.01, 14.17 y 10.05 segundos. En promedio, los algoritmos tienen tiempos de ejecución bastante similares, aunque A* tuvo un desempeño ligeramente más lento en ciertos intentos. Este comportamiento puede deberse a la complejidad adicional que A* maneja al incorporar la heurística, lo que puede aumentar ligeramente el tiempo de procesamiento.

D. Iteraciones

En términos de iteraciones, Dijkstra realizó significativamente más iteraciones que A* en todos los intentos. Los valores fueron 10348, 10774 y 7623 para Dijkstra, mientras que A* realizó solo 2308, 1814 y 2155 iteraciones, respectivamente. Esta diferencia refleja la eficiencia del algoritmo A* en términos de exploración del espacio de búsqueda, lo que le permite llegar a la solución más rápidamente, especialmente en escenarios donde la heurística es efectiva.

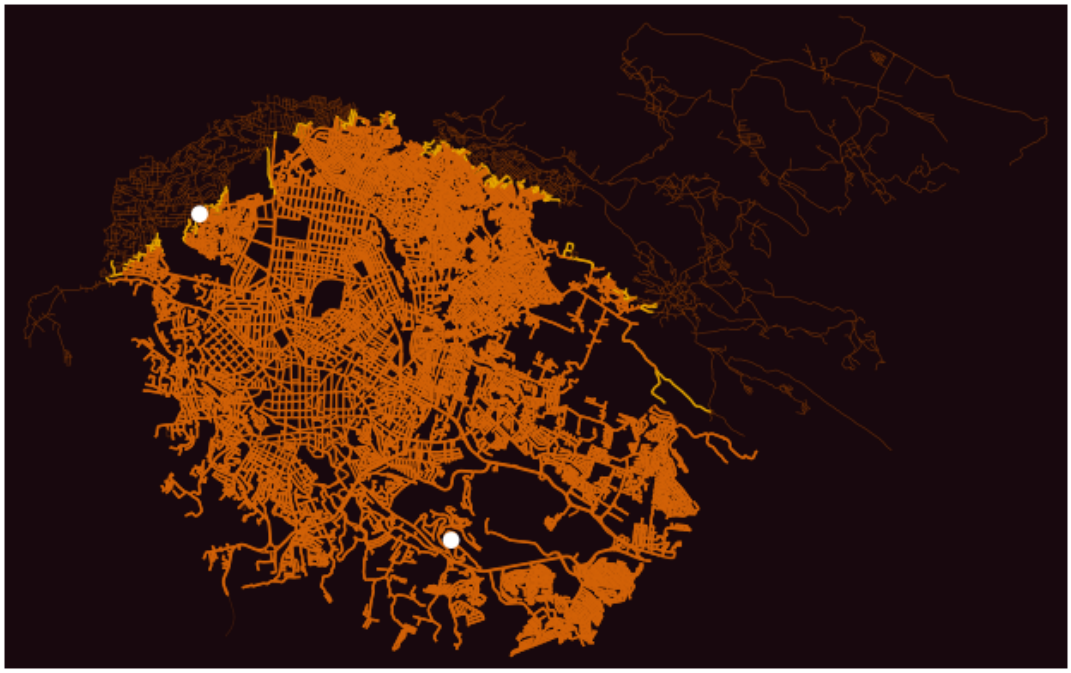

Fig. 1: Todos los nodos recorridos por el algoritmo Dijskstra para llegar a un punto lejano de la ciudad.

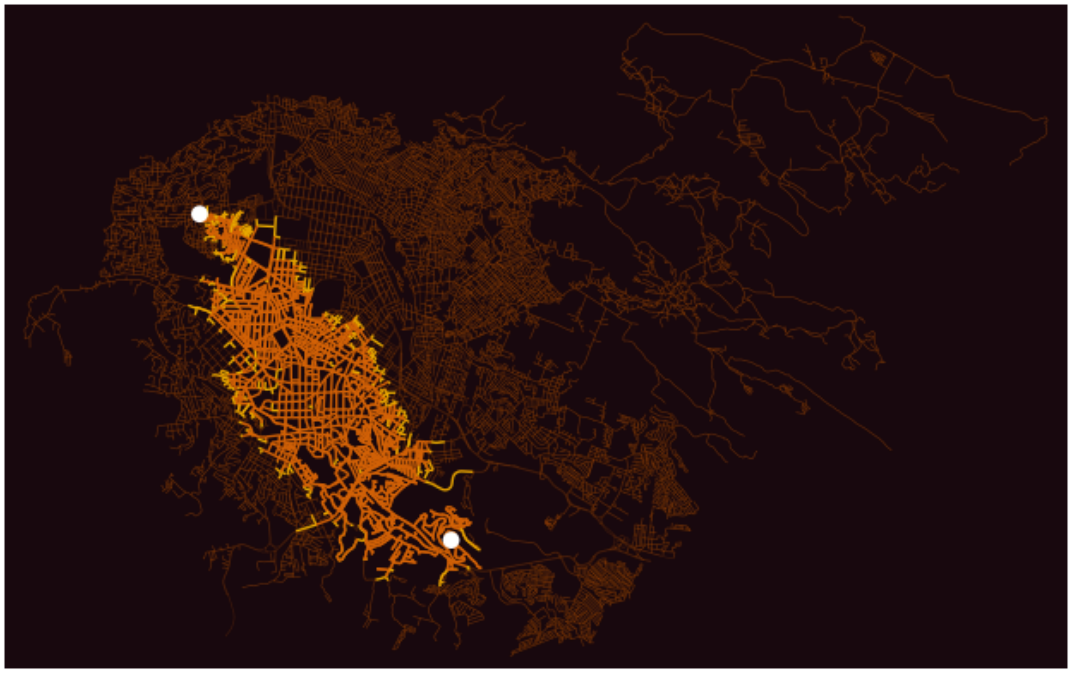

Fig. 2: Todos los nodos recorridos por el algoritmo A* para encontrar el punto objetivo.

Se puede observar por las gráficas el funcionamiento y comportamiento de los dos algoritmos, Dijkstra recorre todos los nodos posibles hasta que encuentra el nodo objetivo, un punto favorable para Dijkstra es que calcula las mejores rutas para todos los nodos que recorre, por lo que no se tiene que ejecutar de nuevo para conocer otra ruta en otro nodo; mientras que A* se enfoca en llegar al objetivo y explora las rutas cercanas desde el nodo inicial hasta el objetivo para poder ver los alrededores y encontrar la mejor ruta.

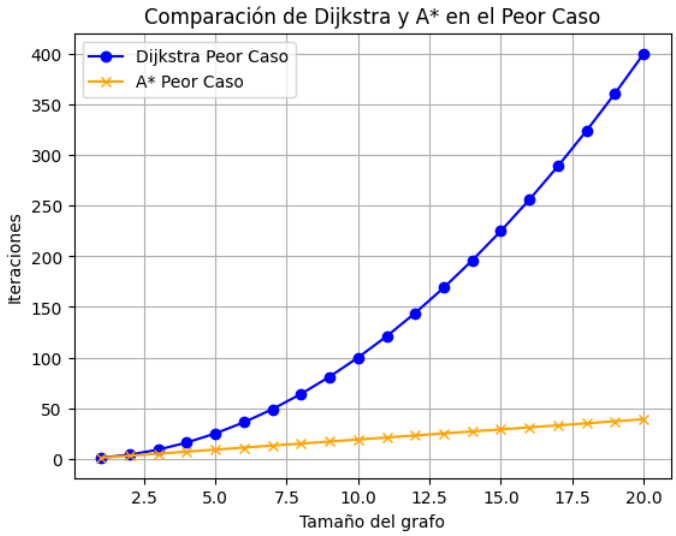

Fig. 3: Peor de los casos o cota superior.

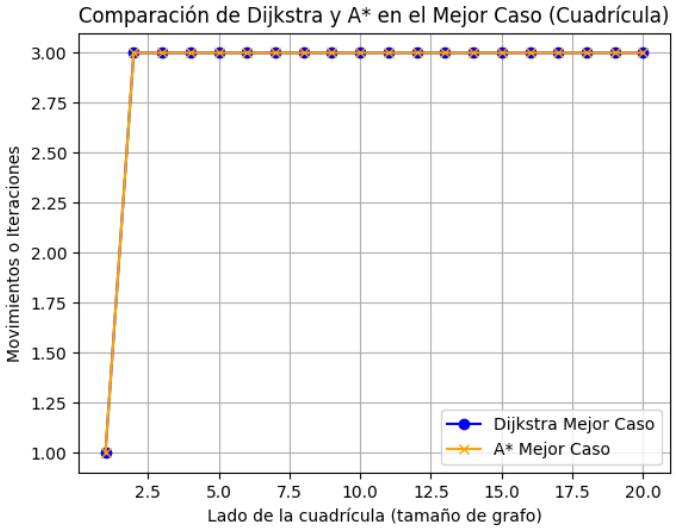

Fig. 4: Mejor de los casos o cota inferior.

En los gráficos se puede observar en el peor de los casos que Dijkstra explora más los nodos y por consecuencia tiene más iteraciones, mientras que A* tiene menos iteraciones a medida que el tamaño del grafo aumenta, es por esto que A* suele ser más rápido, aunque Dijkstra se concentra en encontrar el mejor camino en la mayoría de nodos posibles.//

Mientras que en el mejor caso, aunque el tamaño de los grafos incrementa el resultado es muy constante para ambos algoritmos ya que su objetivo es muy cercano, por lo que en este caso tienen el mismo comportamiento, estoy puede variar con distintos grafos, pero en el mejor de los casos ambos deberían encontrar la mejor ruta con menos iteraciones, mientras más lejos se encuentre el objetivo más se verá la diferencia y el incremento de iteraciones.

V. Conclusiones

La implementación y evaluación de los algoritmos Dijkstra y A* en el contexto de rutas y grafos han revelado comportamientos distintivos en términos de eficiencia, tiempo de ejecución e iteraciones. A lo largo del estudio, se han llegado a las siguientes conclusiones:

A. Eficiencia de Dijkstra

El algoritmo de Dijkstra mostró ser una opción sólida para encontrar las distancias mínimas en grafos, independientemente de la heurística utilizada, ya que no depende de una función heurística, lo que hace que su comportamiento sea más predecible en comparación con A*. Sin embargo, los resultados sugieren que en ciertos casos, su número de iteraciones y tiempo de ejecución pueden ser considerablemente mayores. En los ejemplos proporcionados, Dijkstra necesitó un número significativamente mayor de iteraciones y tiempo en comparación con A* para obtener resultados en rutas similares.

*B. Eficiencia de A**

El algoritmo A* demostró ser más eficiente en términos de iteraciones y tiempo de ejecución en comparación con Dijkstra, especialmente en escenarios donde la función heurística permitió dirigir la búsqueda de manera más efectiva. A pesar de que A* requiere una estimación heurística para optimizar su rendimiento, los resultados indicaron que el algoritmo aprovechó estas estimaciones para reducir significativamente las iteraciones y mejorar la velocidad de ejecución. La diferencia en el tiempo total y las iteraciones entre A* y Dijkstra fue notable, con A* mostrando un rendimiento superior en la mayoría de los casos.

C. Impacto de las Iteraciones y el Tiempo

El análisis también reveló que A* tiende a ser más rápido en rutas con una heurística bien diseñada, mientras que Dijkstra es más uniforme pero menos eficiente en términos de tiempo y iteraciones. En los experimentos realizados, la diferencia en el número de iteraciones entre ambos algoritmos fue considerable, con A* ejecutándose con un número de iteraciones mucho menor, lo que contribuyó a un tiempo de ejecución más corto.

En general, aunque ambos algoritmos son efectivos para resolver problemas de caminos más cortos en grafos, A* se mostró más eficiente, especialmente cuando se dispone de una heurística adecuada. Dijkstra sigue siendo una opción robusta, especialmente cuando no se cuenta con una función heurística confiable. La elección del algoritmo debe basarse en las características del grafo y la disponibilidad de una heurística adecuada, siendo A* la opción preferida cuando se requiere un mejor rendimiento en términos de tiempo y número de iteraciones.

REFERENCES

[1] R. Sedgewick and K. Wayne, *Algorithms*. Addison-wesley professional, 2011.

[2] A. Javaid, "Understanding dijkstra's algorithm," *Available at SSRN 2340905*, 2013.

[3] K. Shanmugam and D. Durgabhavani, "A* star algorithm visualization," 2024.