Optimización de Rutas de Transporte Público Utilizando la Teoría de Grafos y GraphStream

Valentina Vega
Universidad Pedagógica y Tecnológica de Colombia
Email: valentina.vega01@uptc.edu.co

Laura Daniela Guevara
Universidad Pedagógica y Tecnológica de Colombia
Email: laura.guevara02@uptc.edu.co

***Abstract*—Este trabajo presenta el uso de la teoría de grafos para gestionar rutas de transporte público, modelando las paradas como nodos y los recorridos como aristas, através del algoritmo de Dijkstra, se busca optimizar los tiempos de viaje y mejorar la eficiencia en el servicio. La herramienta seleccionada para este proyecto es *GraphStream*, una biblioteca de Java que permite la creación y visualización de grafos dinámicos, esta elección se justifica por su capacidad para manejar grafos dinámicos, su integración con Java y su capacidad para ser utilizada en simulaciones en tiempo real.**

I. Introducción

La gestión eficiente de las rutas en los sistemas de transporte público es uno de los mayores desafíos a los que se enfrentan las ciudades modernas, con el aumento de la población y la creciente demanda de servicios de transporte, es esencial optimizar los recorridos para reducir los tiempos de espera y minimizar los costos operativos, en este contexto, uno de los problemas más críticos es determinar la ruta más corta y eficiente entre dos estaciones, teniendo en cuenta los diferentes recorridos, las conexiones entre las paradas y las características del tráfico, en otras palabras este panorama exige soluciones tecnológicas que permitan optimizar lo problemas ya mencionados, con el fin de mejorar la experiencia del usuario

La teoría de grafos proporciona un marco matemático eficaz para modelar estos problemas, en este modelo, las **paradas de transporte público** se representan como **nodos**, y los **recorridos entre estas paradas** se modelan como **aristas**, cuyos pesos pueden reflejar el tiempo de viaje, la distancia o la carga de tráfico, al usar grafos, es posible aplicar algoritmos de optimización, como el de **Dijkstra**, para encontrar las rutas más cortas entre dos puntos del sistema, evitando así rutas congestionadas y mejorando la eficiencia global del servicio, no obstante, al mismo tiempo esto lleva a optimizar el uso de recursos y reducir el impacto ambiental.[1]

Además, con la integración de tecnologías modernas, es posible modelar y analizar estos grafos en tiempo real, considerando cambios dinámicos en las condiciones del tráfico o la disponibilidad de recursos.

Es así como este trabajo presenta un enfoque para optimizar las rutas de transporte público mediante el uso de la teoría de grafos, implementado en el lenguaje de programación Java y utilizando herramientas como *GraphStream*.

II. Algoritmos de Recorrido en Grafos

En el estudio de los algoritmos de recorrido en grafos, se encuentran varios enfoques fundamentales que permiten explorar las estructuras de un grafo y obtener información relevante, como la identificación de rutas óptimas, la exploración de vecinos o la optimización de caminos. Entre los algoritmos más destacados se incluyen los de recorrido por niveles, profundidad y heurístico, cada uno con aplicaciones particulares en problemas de optimización de redes de transporte público, como la búsqueda de rutas más cortas o la minimización de los tiempos de espera[2].

A. Recorrido por Niveles (BFS - Breadth-First Search)

El recorrido por niveles, conocido también como BFS (Breadth-First Search), es un algoritmo que explora los nodos de un grafo en niveles sucesivos, comenzando desde un nodo inicial y explorando todos los nodos vecinos en el mismo nivel antes de pasar al siguiente. Este algoritmo es ideal para encontrar el camino más corto en un grafo no ponderado, ya que garantiza que la primera vez que se visite un nodo, se habrá encontrado el camino más corto hasta él.

- **Características:**
  - Explora los nodos en niveles sucesivos.
  - Garantiza encontrar el camino más corto en grafos no ponderados.
- **Aplicación en transporte público:**
  - Puede utilizarse para encontrar rutas directas entre paradas sin considerar los tiempos de viaje, pero es útil para identificar la mínima cantidad de paradas.
- **Ventajas:**
  - Eficaz para encontrar rutas de menor cantidad de pasos.
  - Ideal para grafos no ponderados.
- **Desventajas:**
  - No es eficiente para grafos ponderados.

B. Recorrido por Profundidad (DFS - Depth-First Search)

El algoritmo de recorrido por profundidad (DFS - Depth-First Search) explora un grafo comenzando desde un nodo inicial y sigue un camino hasta que no pueda avanzar más, retrocediendo en caso de encontrar un callejón sin salida y explorando nuevos caminos. Este algoritmo es útil para explorar completamente un grafo y es más eficiente que BFS en términos de memoria.[3]

- **Características:**
  - Explora profundamente cada camino antes de retroceder.
  - No garantiza encontrar el camino más corto.
- **Aplicación en transporte público:**
  - Puede ser utilizado para explorar todas las rutas posibles entre dos estaciones, aunque no garantiza la mínima distancia o el tiempo más corto.
- **Ventajas:**
  - Requiere menos memoria que BFS.
  - Puede ser útil para problemas de conectividad o exploración exhaustiva.
- **Desventajas:**
  - No es adecuado para encontrar rutas óptimas en grafos ponderados.

C. Recorrido Heurístico (A* - A Star)

El algoritmo heurístico A* es un algoritmo de búsqueda que utiliza una función de evaluación (o heurística) para guiar el proceso de exploración del grafo, combinando los beneficios de la búsqueda de amplitud de BFS y la búsqueda por profundidad de DFS. A* se utiliza para encontrar rutas óptimas en grafos ponderados, como en los sistemas de transporte, donde cada arista tiene un costo asociado, como el tiempo de viaje o la distancia entre las estaciones[[4].

- **Características:**
  - Utiliza una heurística para estimar el costo de alcanzar el objetivo.
  - Ideal para encontrar el camino más corto en grafos ponderados.
- **Aplicación en transporte público:**
  - Es el algoritmo más adecuado para optimizar rutas en redes de transporte público donde las aristas tienen un costo asociado, como los tiempos de viaje entre paradas.
- **Ventajas:**
  - Garantiza encontrar el camino más corto, si la heurística es adecuada.
  - Muy eficiente para grafos grandes y complejos.
- **Desventajas:**
  - Requiere una heurística adecuada para ser eficiente.
  - Puede ser costoso en términos de tiempo de cómputo si la heurística no está bien optimizada.

D. Comparación de Algoritmos

A continuación, se presenta una tabla comparativa entre los tres algoritmos descritos, destacando sus ventajas y desventajas en el contexto de la optimización de rutas en redes de transporte público.

Algoritmo	Ventajas	
BFS	Encuentra el camino más corto en grafos no ponderados.	No adecuad
DFS	Requiere menos memoria que BFS.	No garantiza e
A*	Garantiza el camino más corto en grafos ponderados.	Requiere

TABLE I
COMPARACIÓN ENTRE ALGORITMOS DE RECORRIDO

III. Herramientas para Gestión de Grafos

En el análisis de herramientas para la gestión de grafos se consideran varias opciones populares en la investigación y la industria.

A. Gephi

Gephi es una plataforma de software de código abierto diseñada para visualizar y analizar redes de grafos, funciona cargando datos de grafos y permite representar visualmente estos grafos en 2D o 3D. La herramienta ofrece múltiples opciones de análisis, como el cálculo de centralidad, detección de comunidades y análisis de caminos más cortos. Se usa principalmente para redes estáticas y la exploración visual de estructuras complejas, finalmente la interfaz permite manipular los grafos de manera interactiva, ajustar parámetros de visualización y aplicar métricas para estudiar sus características.[5]

- **Características:**
  - Visualización interactiva en 2D y 3D.
  - Compatible con formatos como CSV, GEXF y GraphML.
- **Ventajas:**
  - Interfaz intuitiva para explorar redes complejas.
  - Soporte para análisis avanzados de redes.
- **Desventajas:**
  - No se integra directamente con lenguajes de programación.

B. Graphviz

Graphviz es una herramienta especializada en la creación de diagramas de grafos estáticos. Usa el lenguaje de descripción de grafos DOT, donde se definen los nodos y las aristas de un grafo mediante un lenguaje textual, los diagramas generados pueden representar topologías o relaciones jerárquicas, lo que lo hace ideal para ilustrar la estructura de redes de comunicación o flujos de información. Graphviz convierte el código DOT en representaciones gráficas en formatos como PNG, PDF, o SVG. Su funcionamiento es principalmente visual, sin capacidades dinámicas o de análisis en tiempo real.[6]

- **Características:**
  - Generación de gráficos a partir de descripciones en lenguaje DOT.
  - Soporte para grafos dirigidos y no dirigidos.
- **Ventajas:**
  - Alta calidad en la representación gráfica.
  - Facilita la generación automática de visualizaciones.
- **Desventajas:**
  - Limitado en la interacción dinámica en tiempo real.

C. NetworkX

NetworkX es una biblioteca de Python utilizada para la creación, manipulación y análisis de grafos que permite trabajar con grafos dirigidos y no dirigidos y tiene una amplia gama de algoritmos de optimización e investigación sobre grafos, como los de caminos más cortos, centralidad, ciclos, entre otros.

NetworkX no tiene un motor de visualización avanzado integrado, pero puede generar representaciones simples de grafos y se integra fácilmente con otras bibliotecas como Matplotlib para visualizaciones más complejas, el código es bastante flexible, permitiendo al usuario realizar todo tipo de análisis y simulaciones sobre las estructuras de grafos.[7]

- **Características:**
  - Permite trabajar con grafos dirigidos y no dirigidos.
  - Ofrece herramientas para análisis de grafos, como el cálculo de rutas más cortas.
- **Ventajas:**
  - Integra con facilidad con otros paquetes de Python, como NumPy y Matplotlib.
- **Desventajas:**
  - No es tan intuitivo como Gephi para usuarios no técnicos.

D. Cytoscape

Cytoscape es una plataforma de código abierto diseñada para visualizar, analizar y explorar redes complejas, funciona integrando datos provenientes de diversas fuentes, lo que permite crear y analizar redes biológicas, de comunicación, o de cualquier tipo que se pueda representar como un grafo, además ofrece herramientas para el análisis de redes, detección de comunidades, visualización avanzada y tiene la capacidad de integrarse con bases de datos externas para actualizar dinámicamente los datos en las redes, finalmente también permite realizar análisis avanzados como la búsqueda de rutas y la identificación de estructuras clave dentro de las redes.[8]

- **Características:**
  - Análisis y visualización de redes grandes.
  - Integración con bases de datos externas.
- **Ventajas:**
  - Personalización avanzada en la visualización.
  - Soporta scripting para automatización.
- **Desventajas:**
  - Requiere configuraciones adicionales para integrarse con Python o Java.

E. GraphStream

GraphStream es una biblioteca Java para la creación y visualización de grafos dinámicos que se enfoca en representar grafos cuya estructura cambia a lo largo del tiempo, lo que es ideal para aplicaciones como la simulación de tráfico o redes de comunicación, en esta los grafos pueden modificarse en tiempo real, agregando o eliminando nodos y aristas, lo que permite representar redes que varían con el tiempo, incluye algoritmos como Dijkstra, BFS y DFS para realizar análisis de los grafos, además es útil para simulaciones interactivas y dinámicas, y se integra bien con aplicaciones Java, permitiendo la visualización de los resultados en tiempo real [9].

- **Características:**
  - Soporta grafos dinámicos y estáticos.
  - Incluye algoritmos como Dijkstra y BFS.
- **Ventajas:**
  - Visualización en tiempo real.
  - Fácil integración con aplicaciones en Java.
- **Desventajas:**
  - La documentación es limitada en comparación con otras bibliotecas.

IV. Justificación de la elección de GraphStream

Tras considerar las diversas herramientas, se ha seleccionado *GraphStream* debido a las siguientes características que se alinean con los objetivos del proyecto:

- **Integración con Java:** *GraphStream* se integra de manera fluida con Java, lo cual es esencial para el proyecto.

- **Soporte para grafos dinámicos:** Su capacidad para manejar grafos dinámicos es clave, ya que los recorridos de transporte público pueden cambiar dependiendo de las condiciones del tráfico en tiempo real.

- **Visualización en tiempo real:** La visualización en tiempo real es crucial para simular el comportamiento de las rutas de transporte público bajo condiciones variables.

V. Conclusión

La optimización de rutas en redes de transporte público es un desafío complejo que puede abordarse mediante la aplicación de algoritmos de recorrido en grafos, además junto con la selección de una herramienta adecuada, como GraphStream, que permite trabajar de forma eficiente con grafos dinámicos y en tiempo real, será fundamental para la implementación exitosa de un sistema de optimización de rutas en redes de transporte público.

VI. Enlace al video

Enlace drive https://drive.google.com/file/d/1a-Cs53Magj8nnTd-uK8pghIqTHXqfjEf/view?usp=drive_link

Enlace youtube: https://youtu.be/FcYBlui73q8

VII. Repositorio del Proyecto

URL del repositorio en GitHub: https://github.com/Vals2512/Graphs-activity.

VIII. Referencias

1) E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
2) A. Aho and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
3) R. Diestel, *Graph Theory*, 5th ed. Springer-Verlag, 2017.
4) C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, 1982.
5) Gephi Documentation, [Online]. Available: https://gephi.org/users/.
6) Graphviz Documentation, [Online]. Available: https://graphviz.gitlab.io/documentation/.
7) NetworkX Documentation, [Online]. Available: https://networkx.github.io/documentation/stable/.
8) Cytoscape Documentation, [Online]. Available: http://cytoscape.org/.
9) GraphStream Documentation, [Online]. Available: https://graphstream-project.org/.