Facultad de Ciencias Económicas y Estadística

Universidad Nacional de Rosario

Anteproyecto

[Acá va el título de la tesina]

Director: Iván Millanes
Codirector: Diego Marfetán Molina
Alumna: Valentina Salvarezza

Índice

1 Introducción

En la actualidad, cuando una persona necesita desplazarse de un punto a otro, es común recurrir a aplicaciones como Google Maps. Estas herramientas han transformado la manera en que se planifican los desplazamientos, proporcionando información precisa sobre las rutas más convenientes, ya sea caminando, en automóvil, en bicicleta o utilizando otros medios de transporte. Además de indicar el recorrido a seguir, muchas de estas aplicaciones estiman el tiempo de llegada y consideran factores como el tráfico o las condiciones meteorológicas para ofrecer mejores alternativas.

Sin embargo, surgen algunas preguntas fundamentales: ¿cómo funcionan realmente estas aplicaciones? ¿Cómo son capaces de calcular la ruta más corta o más rápida en función de las preferencias del usuario? Las respuestas se encuentran en la teoría de grafos, una rama de las matemáticas y la informática que permite modelar relaciones y estructuras en diversas áreas, como redes de comunicación, análisis de redes sociales y optimización de sistemas logísticos. En el contexto de la planificación de rutas, el problema se modela mediante un grafo ponderado, donde los nodos representan intersecciones y las aristas, los caminos que los conectan. A través de este modelo es posible calcular las rutas más eficientes, evaluando cada trayecto en función de costos asociados, como la distancia o el tiempo de viaje, e identificando la opción óptima que minimice dichos costos.

Para resolver este problema, se emplea el algoritmo de Dijkstra, desarrollado por Edsger W. Dijkstra en 1956 y publicado en 1959. Este algoritmo, ampliamente utilizado en teoría de grafos, permite encontrar el camino más corto entre dos nodos cuando los pesos de las aristas son no negativos. Su eficiencia y robustez lo convierten en una herramienta clave en diversas aplicaciones, como sistemas de navegación, redes informáticas y planificación logística. Uno de los servidores que implementa este algoritmo es OSRM (Open Source Routing Machine), un enrutador de código abierto diseñado para procesar datos provenientes de OpenStreetMap (OSM). OSRM, al integrarse con OSM, calcula rutas optimizadas en función de distintos criterios, como el medio de transporte o restricciones en el camino. Su capacidad para manejar grandes volúmenes de datos y ofrecer respuestas rápidas lo convierte en una solución ampliamente utilizada en aplicaciones de movilidad, logística y optimización de rutas.

En este trabajo, se desarrolla una aplicación web utilizando Shiny en RStudio. Shiny es una herramienta interactiva que permite crear aplicaciones web dinámicas y reactivas directamente desde R. Esta aplicación está diseñada para calcular recorridos óptimos entre diferentes puntos

de la ciudad de Rosario, permitiendo al usuario seleccionar intersecciones como puntos de origen y destino. A partir de esta selección, se genera el recorrido más corto o el más rápido, según la preferencia del usuario. Comprender cómo se estructuran y procesan estos datos no solo permite analizar en profundidad el funcionamiento de estas herramientas, sino también explorar aplicaciones en movilidad urbana, planificación del transporte público y optimización logística.

2 Objetivos

2.1 Objetivo General

El objetivo principal de esta tesina es desarrollar una herramienta para la optimización de rutas en la ciudad de Rosario, basada en la aplicación del algoritmo de Dijkstra. La implementación se realiza a través de una Shiny App en RStudio, que permite calcular recorridos eficientes para vehículos considerando criterios de distancia y tiempo de viaje.

2.2 Objetivos Específicos:

- Analizar los datos geográficos representados por nodos (intersecciones) y aristas (calles y caminos), que constituyen un tipo de dato no convencional, nunca antes tratado en la carrera, para evaluar su adecuación y optimización en el cálculo de rutas.
- Desarrollar e implementar el algoritmo de Dijkstra y su extensión Multi Level Dijkstra (MLD) como métodos de optimización para el cálculo de rutas en grafos ponderados por distancia y tiempo de viaje. Esto permite generar dos tipos de recorridos: el más corto y el más rápido.
- Crear una Shiny App en RStudio para visualizar las rutas optimizadas para vehículos, utilizando un servidor de OSRM desplegado localmente. Este servidor es crucial para realizar el cálculo de las rutas, ya que procesa los datos geográficos de Rosario y genera los recorridos más cortos y rápidos, los cuales se muestran a través de una interfaz interactiva.

3 Metodología

3.1 Teoría de Grafos

La teoría de grafos es una rama central de las matemáticas y la informática que se dedica al estudio de las relaciones entre objetos representados por nodos o vértices, y las aristas, que son las conexiones entre ellos. Un grafo G se define formalmente como $G = (V, E)$, donde V es un conjunto no vacío de vértices y E es el conjunto de aristas que conectan pares de vértices. Estos modelos resultan fundamentales en la optimización de rutas, dado que permiten representar intersecciones como nodos y calles como aristas, facilitando el análisis de recorridos.

Algunos de los grafos que se utilizan para modelar redes viales son los siguientes:

- **Grafo simple:** Las aristas no tienen dirección, por lo que el recorrido puede hacerse en ambos sentidos. Este tipo de grafo se utiliza cuando se desea representar conexiones bidireccionales, como calles donde se puede circular en ambos sentidos. Un caso de aplicación se da cuando el desplazamiento es a pie, ya que caminando es posible combinar ambos sentidos de la calle para encontrar el camino más corto en términos de cantidad de aristas.
- **Grafo dirigido:** En este caso, las aristas tienen una orientación específica, representada como un par ordenado de vértices (v, w). Esto implica que la conexión desde v hacia w no necesariamente permite el recorrido inverso, lo que lo hace adecuado para modelar calles con sentido único.
- **Grafo ponderado:** Cada arista lleva asociado un peso $p(e)$, donde e corresponde al peso de la arista, que puede representar distancia, tiempo de viaje u otro costo. La longitud de un camino se mide como la suma de los pesos de sus aristas, lo que permite modelar tanto el recorrido más corto como el más rápido.

En síntesis, la elección del tipo de grafo determina cómo se representan las restricciones y características de la red vial. En el presente trabajo, se emplean principalmente grafos dirigidos y ponderados, donde la ponderación se realiza por distancia o por tiempo, según el criterio de optimización considerado.

3.2 Algoritmo de Dijkstra

El algoritmo de Dijkstra, propuesto por Edsger W. Dijkstra en 1956, es un método ampliamente utilizado para determinar el camino de costo mínimo entre un nodo de origen y los demás nodos de un grafo ponderado. En estos grafos, cada arista tiene un peso que puede representar distancia, tiempo u otro tipo de costo, y se asume que estos pesos son no negativos. Este algoritmo resulta fundamental en sistemas de navegación y optimización de rutas, ya que permite calcular caminos mínimos de manera rápida y precisa.

La idea central del algoritmo consiste en avanzar desde un nodo de origen hacia los nodos adyacentes, seleccionando las rutas de menor costo en cada paso, hasta llegar al destino. Para ello, a cada nodo se le asigna un valor $D(v)$, que es una función que representa el costo mínimo desde el nodo de origen hasta ese nodo v. Inicialmente, para el nodo de origen o, su peso se define como $D(o) = 0$. Para todos los demás nodos, el valor $D(v)$ se establece como ∞, lo que indica que aún no se ha calculado el costo mínimo desde el nodo de origen.

Durante la ejecución del algoritmo, los nodos se dividen en dos categorías:

- **Etiquetas temporales:** Son aquellas etiquetas que pueden actualizarse a medida que se encuentra un camino más corto hacia el nodo. Estas etiquetas representan posibles costos mínimos que aún no han sido confirmados de manera definitiva.
- **Etiquetas permanentes:** Indican que se ha determinado el menor costo posible hacia ese nodo. Una vez que un nodo tiene una etiqueta permanente, su costo no se actualiza más, ya que se considera que se ha encontrado el camino más corto definitivo hacia él.

El procedimiento del algoritmo se desarrolla de la siguiente manera:

1. **Inicialización:** Se establece el costo mínimo desde el nodo de origen hacia sí mismo $D(o) = 0$, y para todos los demás nodos $D(t) = \infty$, ya que no se conoce el costo hacia ellos. Donde t pertenece al conjunto T, el cual se define como el conjunto de todos los nodos no visitados, es decir, aquellos nodos cuya distancia mínima desde el origen aún no ha sido confirmada.
2. **Selección del nodo con menor costo:** Se selecciona el nodo u del conjunto T con el valor más bajo de $D(u)$, es decir, el nodo cuyo costo desde el origen es el mínimo entre los nodos no visitados.

3. **Cambiar nodo temporal a permanente:** El nodo u se elimina del conjunto T porque su costo ha sido confirmado y su etiqueta se convierte en permanente.

4. **Actualización de los costos:** Para cada nodo vecino t de u, si el costo de llegar a t desde el origen pasando por u es menor que el valor actual de $D(t)$, se actualiza el valor de $D(t)$ utilizando la fórmula: $D(t) = min\{D(t), D(u) + p(u,t)\}$, donde $p(u,t)$ representa el peso de la arista que conecta u con t.

5. Repetir hasta encontrar el camino mínimo: Este proceso se repite hasta que todos los nodos hayan sido visitados o el nodo destino haya sido alcanzado.

El algoritmo garantiza que, al asignar una etiqueta permanente a un nodo, se ha determinado el costo mínimo desde el origen hacia dicho nodo. Esto se debe a que, durante su ejecución, se evalúan todas las rutas posibles para llegar a cada nodo, asegurando la solución óptima. Sin embargo, este enfoque resulta computacionalmente costoso cuando se aplica al cálculo de recorridos en ciudades grandes, como Rosario. Por este motivo, en contextos de mayor escala, se utiliza una extensión del algoritmo llamada Multi-Level Dijkstra.

3.3 Multi-Level Dijkstra (MLD)

El Multi-Level Dijkstra (MLD) es una extensión del algoritmo de Dijkstra diseñada para optimizar la eficiencia en el cálculo de rutas en redes viales grandes, como las que se encuentran en ciudades extensas. Su objetivo principal es reducir la complejidad computacional y el tiempo de procesamiento al calcular rutas sobre grandes bases de datos geográficos.

El enfoque del MLD se basa en una estructura jerárquica de niveles:

- **Nivel 0:** el grafo original se divide en pequeñas celdas que representan barrios o vecindarios, incluyendo todas las calles e intersecciones internas.
- **Nivel 1:** varias celdas de nivel 0 se agrupan en unidades más grandes, como distritos, reduciendo la complejidad al analizar recorridos entre distintas zonas de la ciudad.
- **Nivel 2 y superiores:** los distritos se combinan en macrozonas que abarcan áreas extensas, por ejemplo, Norte, Sur, Este y Oeste.

Durante la fase de preprocesamiento, el algoritmo identifica los nodos frontera, es decir, las intersecciones que conectan diferentes celdas, y calcula previamente los caminos más cortos entre ellos. Con esta información se construye un grafo reducido (overlay graph) que actúa como un conjunto de atajos entre celdas, permitiendo que el cálculo de rutas se concentre únicamente en las áreas relevantes.

El procedimiento del Multi-Level Dijkstra (MLD) funciona de la siguiente manera:

- Cuando el origen y destino se encuentran dentro de la misma celda de nivel 0, la búsqueda se realiza directamente sobre el subgrafo local mediante el algoritmo de Dijkstra, sin necesidad de utilizar MLD.
- Cuando el origen y destino están en celdas distintas, se utiliza un grafo reducido de niveles superiores para realizar un "salto" entre regiones. Luego, se desciende a niveles más detallados hasta llegar al nivel más bajo y reconstruir el trayecto completo, calle por calle.

Este enfoque permite mantener la precisión en la ruta óptima, mientras se reduce significativamente la cantidad de nodos a explorar, lo que resulta crucial para aplicaciones de movilidad urbana que manejan grandes volúmenes de datos geográficos, como los provenientes de OpenStreetMap (OSM).

En resumen, el MLD combina la exhaustividad del algoritmo de Dijkstra con una estructura jerárquica eficiente, optimizando tanto el tiempo de cálculo como la gestión de rutas en redes urbanas extensas.

3.4 Implementación y configuración de OSRM para la ciudad de Rosario

Open Source Routing Machine (OSRM) es una herramienta de enrutamiento de código abierto que utiliza datos geográficos de OpenStreetMap (OSM) para calcular rutas óptimas en redes viales. Implementa el algoritmo de Dijkstra y una extensión optimizada llamada Multi-Level Dijkstra, lo que le permite generar rutas rápidas y precisas incluso con grandes volúmenes de datos. OSRM se utiliza principalmente en aplicaciones de navegación y en sistemas donde se requiere cálculo eficiente de rutas.

3.4.1 Obtención de los datos

Los datos geográficos se obtienen de OpenStreetMap mediante la plataforma BBbike, que permite descargar áreas específicas en formato `.osm.pbf`. Este formato es requerido por OSRM ya que está comprimido y estructurado de manera eficiente para ser procesado rápidamente durante la fase de preparación de datos.

Se descargan los datos correspondientes a toda la ciudad de Rosario, asegurando que solo se incluyan los elementos (intersecciones y calles) relevantes para el cálculo de rutas. De esta forma, se evita el procesamiento de información irrelevante y se optimiza el rendimiento del sistema.

3.4.2 Cálculo de rutas

Aunque OSRM ofrece un servidor público para el cálculo de rutas, este servicio tiene limitaciones importantes: si se superan ciertos volúmenes de solicitudes (requests), puede saturarse o incluso rechazar peticiones. Debido a esta restricción, y para garantizar un funcionamiento estable durante las pruebas y el desarrollo, se opta por implementar servidores OSRM locales.

Como no es posible usar un mismo servidor de OSRM para calcular dos tipos de recorridos distintos (camino más corto y camino más rápido), se configuran dos servidores locales independientes. Para lograr esto, se modifica el archivo de perfil de enrutamiento `car.lua`, que define cómo se interpretan los datos del mapa y qué criterio se prioriza al generar una ruta. La configuración se realiza de la siguiente manera:

- **Servidor para el camino más corto:** se establece `weight_name = 'distance'` en `car.lua`, lo que indica que debe priorizar recorridos con la menor distancia posible.
- **Servidor para el camino más rápido:** se configura `weight_name = 'duration'` en `car.lua`, lo que permite calcular rutas optimizadas según el tiempo estimado de viaje.

Esta estrategia permite generar ambos tipos de rutas de forma precisa y controlada, sin depender del servidor público de OSRM y sus restricciones operativas.

3.4.3 Despliegue en la nube

Una vez construidos los servidores OSRM locales, es necesario desplegarlos en la nube. Esto se debe a que, al publicar la aplicación en shinyapps.io, los servidores deben estar accesibles de forma remota. Si permanecen en la red local, la aplicación no puede comunicarse con ellos una vez desplegada.

Por este motivo, ambos servidores se despliegan en la nube utilizando los servicios de DonWeb, una plataforma de hosting que permite contratar servidores virtuales (VPS) con acceso remoto. Esto garantiza que los servicios de enrutamiento estén disponibles desde cualquier ubicación.

Este despliegue es clave para hacer posible la construcción de la aplicación Shiny. Al contar con los servidores OSRM en la nube, se vuelve posible conectarlos a la app y habilitar una funcionalidad central: generar recorridos según el criterio seleccionado.